BASH Cheat Sheet

2017 ICOS Big Data Summer Camp

Most BASH commands

- follow the pattern `$ [command] [options] [input] [output]`
- tell you how to use them if you type `$ [command] --help`
- have a manual file with more info `$ man [command]`
- are explained with examples if you google them “bash [command] example”

Command	Description	Quit	Example
`!!`	Repeat the previous command.		
`cat`	Concatenate. Takes the contents of a file and puts them on the end of something else (your screen, another file, etc.)	`[ctrl]+C`	`cat file.txt`
`cd`	Change Directory. Move from one folder (directory) to another.		`cd my_folder/data`
`cp`	Copy. Make a copy of a file. See also: mv.	`[ctrl]+C`	`cp original.html copy.html`
`diff`	Difference. Print a list of all lines that are different between two files.	`[ctrl]+C`	`diff old.csv new.csv`
`echo`	Echo. Repeat whatever I type next.		`echo "Hello, World!"`
`emacs`	Editor Macros. Program for editing files. Advanced users. See also: vi, nano, pico.		
`find`	Find. Search for files that match some criteria (size, date modified, name, type, and more).	`[ctrl]+C`	`find . -name "*.html" -size +100k`
`grep`	Search for lines of text that match a pattern and print them (similar to [ctrl]+F or [cmd]+F). See also: sed.	`[ctrl]+C`	`grep “href” kitten.html`
`head`	Print just the top (head) of a file. See also: tail.	`[ctrl]+C`	`head long_file.txt`
`htop`	Hisham Table of Processes. Like "top", but with more information and colors.	`[ctrl]+C`	`htop`
`ll`	List Long. The same as "ls -l". Will show the size, owner, date, and permissions for all files in the current directory.	`[ctrl]+C`	`ll -h`
`ls`	List files in the current directory.	`[ctrl]+C`	`ls`
`man`	Manual. Show the manual entry for a command to see how to use it and what the options are. (Use arrow keys to scroll.)	`Q`	`man cat`
`mkdir`	Make Directory. Create a new directory (folder). See also: rmdir.		`mkdir new_folder`
`mv`	Move a file or directory. See also: cp.	`[ctrl]+C`	`mv file.txt subfolder/file.txt`
`nano`	Same as "pico" but released as free software.	`[ctrl]+X`	`nano my_code.py`
`pico`	Pine Composer. Very simple program for editing files in the terminal. See also: vi, nano, emacs.	`[ctrl]+X`	`pico my_code.py`

Command	Description	Quit	Example
`pwd`	Print Working Directory. Show the full path of what directory (folder) you are currently in.		`pwd`
`rm`	Remove. Deletes the specified file(s). Does not send things to a trash folder. They are gone forever.	`[ctrl]+C`	`rm unwanted_file.doc`
`rmdir`	Remove Directory. Deletes a specified directory/folder. See also: mkdir.	`[ctrl]+C`	`rmdir unwanted_directory`
`script`	Make a record of everything that I type and everything that appears in my terminal until I type "exit." Then save that as a file.	`"exit"`	
`sed`	Stream Editor. The sed command can do a lot, but it's most useful function is find and replace in text. See also: grep.	`[ctrl]+C`	`sed 's/dog/cat/g' dog.txt > cat.txt`
`split`	Splits a file into multiple smaller files. See also cat, which can put them back together.	`[ctrl]+C`	`split big_file.csv`
`ssh`	Secure Shell. Connect to a remote server's command line.	`"exit"`	`ssh my.server.umich.edu`
`tail`	Print just the bottom of a file. See also: head.	`[ctrl]+C`	`tail long_file.txt`
`top`	Table Of Processes. Shows running processes memory use. Like WIndows system monitor or Mac activity monitor. See also: htop.	`[ctrl]+C`	`top`
`uname`	Unix Name. Print the name and versio n of my operating system.		`uname -a`
`vi`	Visual (line editor). A program for editing files in the terminal. Intermediate and advanced users. See also: pico, nano, emacs.	`[esc]+[:]+Q`	`vi my_code.py`
`wc`	Word Count. Count many lines, words, and characters are in something.	`[ctrl]+C`	`wc essay.txt`
`wget`	Web Get. Download something from an internet URL.	`[ctrl]+C`	`wget bbc.co.uk`

Symbol	Use
`*`	Wildcard. Select everything. Can be combined with other characters, e.g. "`*.txt`" would match all files ending in "`.txt`" and "`ls *.txt`" will list the files that end in "`.txt`".
`>`	Overwrite. Take the output of the argument to the left and use it to replace the contents of what is on the right. E.g. "`cat updates.txt > latest.txt`" will replace whatever is in 'latest.txt' with whatever is in 'updates.txt'.
`>>`	Append. Take the output of the argument to the left and add it to end end of what is on the right. E.g. "`cat updates.txt >> all.txt`" will add whatever is in 'updates.txt' to the end of 'all.txt' after what is already in there.
`|`	Pipe (usually above the [enter] key). Use the output of the command to the left as input for the command to the right. E.g. in order to count the files in a directory, you can type "`ls | wc -l`". `ls` outputs a list of files, one per line. That list is sent ("piped") to the word count utility with the "`-l`" option to count lines. The result is the count of files.
`;`	End previous command, begin a new one. E.g. "`echo "We're in"; pwd`" would first print the words "We're in" and then it would print the path of the current working directory.

Bash Cheat Sheet

By John Stowers

This file contains short tables of commonly used items in this shell. In most cases the information applies to both the Bourne shell (sh) and the newer bash shell.

Tests (for ifs and loops) are done with [ ] or with the test command.

Checking files:

```
-r file     Check if file is readable.
-w file     Check if file is writable.
-x file     Check if we have execute access to file.
-f file     Check if file is an ordinary file (as opposed to a directory, a device special file, etc.)
-s file     Check if file has size greater than 0.
-d file     Check if file is a directory.
-e file     Check if file exists.  Is true even if file is a directory.
```

Example:

```
if [ -s file ]
then
    #such and such
fi
```

Checking strings:

```
s1 = s2     Check if s1 equals s2.
s1 != s2    Check if s1 is not equal to s2.
-z s1       Check if s1 has size 0.
-n s1       Check if s2 has nonzero size.
s1          Check if s1 is not the empty string.
```

Example:

```
if [ $myvar = "hello" ] ; then
echo "We have a match"
fi
```

Checking numbers:
Note that a shell variable could contain a string that represents a number. If you want to check the numerical value use one of the following:

```
n1 -eq n2      Check to see if n1 equals n2.
n1 -ne n2      Check to see if n1 is not equal to n2.
n1 -lt n2      Check to see if n1 < n2.
n1 -le n2      Check to see if n1 <= n2.
n1 -gt n2      Check to see if n1 > n2.
n1 -ge n2      Check to see if n1 >= n2.
```

Example:

```
if [ $# -gt 1 ]
then
    echo "ERROR: should have 0 or 1 command-line parameters"
fi
```

Boolean operators:

```
!     not
-a    and
-o    or
```

Example:

```
if [ $num -lt 10 -o $num -gt 100 ]
then
    echo "Number $num is out of range"
elif [ ! -w $filename ]
```

```
then
    echo "Cannot write to $filename"
fi
```

Note that ifs can be nested. For example:

```
if [ $myvar = "y" ]
then
    echo "Enter count of number of items"
    read num
    if [ $num -le 0 ]
    then
        echo "Invalid count of $num was given"
    else
#... do whatever ...
fi
fi
```

The above example also illustrates the use of read to read a string from the keyboard and place it into a shell variable. Also note that most UNIX commands return a true (nonzero) or false (0) in the shell variable status to indicate whether they succeeded or not. This return value can be checked. At the command line echo $status. In a shell script use something like this:

```
if grep -q shell bshellref
then
    echo "true"
else
    echo "false"
fi
```

Note that -q is the quiet version of grep. It just checks whether it is true that the string shell occurs in the file bshellref. It does not print the matching lines like grep would otherwise do.

I/O Redirection:

```
pgm > file     Output of pgm is redirected to file.
pgm < file     Program pgm reads its input from file.
pgm >> file    Output of pgm is appended to file.
pgm1 | pgm2    Output of pgm1 is piped into pgm2 as the input to pgm2.
n > file       Output from stream with descriptor n redirected to file.
n >> file      Output from stream with descriptor n appended to file.
n >& m         Merge output from stream n with stream m.
n <& m         Merge input from stream n with stream m.
<< tag         Standard input comes from here through next tag at start of line.
```

Note that file descriptor 0 is normally standard input, 1 is standard output, and 2 is standard error output.

Shell Built-in Variables:

```
$0             Name of this shell script itself.
$1             Value of first command line parameter (similarly $2, $3, etc)
$#             In a shell script, the number of command line parameters.
$*             All of the command line parameters.
$-             Options given to the shell.
$?             Return the exit status of the last command.
$$             Process id of script (really id of the shell running the script)
```

Pattern Matching:

```
*              Matches 0 or more characters.
?              Matches 1 character.
[AaBbCc]       Example: matches any 1 char from the list.
[^RGB]         Example: matches any 1 char not in the list.
[a-g]          Example: matches any 1 char from this range.
```

Quoting:

```
\c             Take character c literally.
`cmd`          Run cmd and replace it in the line of code with its output.
"whatever"     Take whatever literally, after first interpreting $, `...`, \
'whatever'     Take whatever absolutely literally.
```

Example:

```
match=`ls *.bak`        #Puts names of .bak files into shell variable match.
echo \*                 #Echos * to screen, not all filename as in:  echo *
echo '$1$2hello'        #Writes literally $1$2hello on screen.
echo "$1$2hello"        #Writes value of parameters 1 and 2 and string hello.
```

Grouping:
Parentheses may be used for grouping, but must be preceded by backslashes since parentheses normally have a different meaning to the shell (namely to run a command or commands in a subshell). For example, you might use:

```
if test \( -r $file1 -a -r $file2 \) -o \( -r $1 -a -r $2 \)
then
    #do whatever
fi
```

Case statement:
Here is an example that looks for a match with one of the characters a, b, c. If $1 fails to match these, it always matches the * case. A case statement can also use more advanced pattern matching.

```
case "$1" in
    a) cmd1 ;;
    b) cmd2 ;;
    c) cmd3 ;;
    *) cmd4 ;;
esac
```

Loops:
Bash supports loops written in a number of forms,

```
for arg in [list]
do
    echo $arg
done

for arg in [list] ; do
    echo $arg
done
```

You can supply `[list]` directly

```
NUMBERS="1 2 3"
for number in `echo $NUMBERS`
do
  echo $number
done

for number in $NUMBERS
do
  echo -n $number
done

for number in 1 2 3
do
  echo -n $number
done
```

If `[list]` is a glob pattern then bash can expand it directly, for example:

```
for file in *.tar.gz
do
    tar -xzf $file
done
```

You can also execute statements for `[list]`, for example:

```
for x in `ls -tr *.log`
do
   cat $x &gt;&gt; biglog
done
```

Shell Arithmetic:
In the original Bourne shell arithmetic is done using the expr command as in:

```
result=`expr $1 + 2`
result2=`expr $2 + $1 / 2`
result=`expr $2 \* 5`               #note the \ on the * symbol
```

With bash, an expression is normally enclosed using [ ] and can use the following operators, in order of precedence:

```
* / %       (times, divide, remainder)
+ -         (add, subtract)
< > <= >=   (the obvious comparison operators)
== !=       (equal to, not equal to)
&&          (logical and)
||          (logical or)
=           (assignment)
```

Arithmetic is done using long integers.
Example:

```
result=$[$1 + 3]
```

In this example we take the value of the first parameter, add 3, and place the sum into result.

Order of Interpretation:
The bash shell carries out its various types of interpretation for each line in the following order:

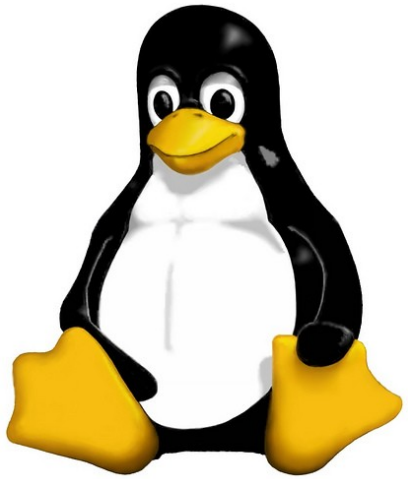
```
brace expansion         (see a reference book)
~ expansion             (for login ids)
parameters              (such as $1)
variables               (such as $var)
command substitution    (Example:  match=`grep DNS *` )
arithmetic              (from left to right)
word splitting
pathname expansion      (using *, ?, and [abc] )
```

Other Shell Features:

```
$var           Value of shell variable var.
${var}abc      Example: value of shell variable var with string abc appended.
#              At start of line, indicates a comment.
var=value      Assign the string value to shell variable var.
cmd1 && cmd2   Run cmd1, then if cmd1 successful run cmd2, otherwise skip.
cmd1 || cmd2   Run cmd1, then if cmd1 not successful run cmd2, otherwise skip.
cmd1; cmd2     Do cmd1 and then cmd2.
cmd1 & cmd2    Do cmd1, start cmd2 without waiting for cmd1 to finish.
(cmds)         Run cmds (commands) in a subshell.
```

freeworld.posterous.com

Linux Bash Shell Cheat Sheet



(works with about every distribution, except for apt-get which is Ubuntu/Debian exclusive)

Legend:

Everything in “<>” is to be replaced, ex: <fileName> --> iLovePeanuts.txt
Don't include the '=' in your commands
'..' means that more than one file can be affected with only one command ex: rm file.txt file2.txt movie.mov .. ..

Linux Bash Shell Cheat Sheet

Basic Commands

Basic Terminal Shortcuts

CTRL L = Clear the terminal
CTRL D = Logout
SHIFT Page Up/Down = Go up/down the terminal
CTRL A = Cursor to start of line
CTRL E = Cursor the end of line
CTRL U = Delete left of the cursor
CTRL K = Delete right of the cursor
CTRL W = Delete word on the left
CTRL Y = Paste (after CTRL U,K or W)
TAB = auto completion of file or command
CTRL R = reverse search history
!! = repeat last command
CTRL Z = stops the current command (resume with fg in foreground or bg in background)

Basic Terminal Navigation

ls -a = list all files and folders
ls <folderName> = list files in folder
ls -lh = Detailed list, Human readable
ls -l *.jpg = list jpeg files only
ls -lh <fileName> = Result for file only

cd <folderName> = change directory
if folder name has spaces use “ “
cd / = go to root
cd .. = go up one folder, tip: ../../../

du -h: Disk usage of folders, human readable
du -ah: “ “ “ files & folders, Human readable
du -sh: only show disc usage of folders

pwd = print working directory

man <command> = shows manual (RTFM)

Basic file manipulation

cat <fileName> = show content of file
(less, more)
head = from the top
-n <#oflines> <fileName>

tail = from the bottom
-n <#oflines> <fileName>

mkdir = create new folder
mkdir myStuff ..
mkdir myStuff/pictures/ ..

cp image.jpg newimage.jpg = copy and rename a file
cp image.jpg <folderName>/ = copy to folder
cp image.jpg folder/sameImageNewName.jpg
cp -R stuff otherStuff = copy and rename a folder
cp *.txt stuff/ = copy all of *<file type> to folder

mv file.txt Documents/ = move file to a folder
mv <folderName> <folderName2> = move folder in folder
mv filename.txt filename2.txt = rename file
mv <fileName> stuff/newfileName
mv <folderName>/ .. = move folder up in hierarchy

rm <fileName> .. = delete file (s)
rm -i <fileName> .. = ask for confirmation each file
rm -f <fileName> = force deletion of a file
rm -r <foldername>/ = delete folder

touch <fileName> = create or update a file

ln file1 file2 = physical link
ln -s file1 file2 = symbolic link

Linux Bash Shell Cheat Sheet

Basic Commands

Researching Files

The slow method (sometimes very slow):

```
locate <text> = search the content of all the files
locate <fileName> = search for a file
sudo updatedb = update database of files

find = the best file search tool (fast)
find -name “<fileName>”
find -name “text” = search for files who start with the word text
find -name “*text” = “ “ “ “ end “ “ “ “
```

Advanced Search:

```
Search from file Size (in ~)
      find ~ -size +10M = search files bigger than.. (M,K,G)

Search from last access
      find -name “<filetype>” -atime -5
            ('-' = less than, '+' = more than and nothing = exactly)

Search only files or directory’s
      find -type d --> ex: find /var/log -name "syslog" -type d
      find -type f = files

More info: man find, man locate
```

Extract, sort and filter data

```
grep <someText> <fileName> = search for text in file
      -i = Doesn't consider uppercase words
      -I = exclude binary files
grep -r <text> <folderName>/ = search for file names
      with occurrence of the text
```

With regular expressions:

```
grep -E ^<text> <fileName> = search start of lines
with the word text
grep -E <0-4> <fileName> =shows lines containing numbers 0-4
grep -E <a-zA-Z> <fileName> = retrieve all lines
with alphabetical letters

sort = sort the content of files
sort <fileName> = sort alphabetically
sort -o <file> <outputFile> = write result to a file
sort -r <fileName> = sort in reverse
sort -R <fileName> = sort randomly
sort -n <fileName> = sort numbers

wc = word count
wc <fileName> = nbr of line, nbr of words, byte size
      -l (lines), -w (words), -c (byte size), -m
      (number of characters)

cut = cut a part of a file
-c --> ex: cut -c 2-5 names.txt
      (cut the characters 2 to 5 of each line)
-d (delimiter)        (-d & -f good for .csv files)
-f (# of field to cut)

more info: man cut, man sort, man grep
```

Linux Bash Shell Cheat Sheet

Basic Commands

Time settings

date = view & modify time (on your computer)

View:
date “+%H” --> If it's 9 am, then it will show 09
date “+%H:%M:%Ss” = (hours, minutes, seconds)
%Y = years
Modify:
MMDDhhmmYYYY
Month | Day | Hours | Minutes | Year

sudo date 031423421997 = March 14th 1997, 23:42

Execute programs at another time

use 'at' to execute programs in the future

Step 1, write in the terminal: at <timeOfExecution> ENTER
ex --> **at 16:45** or **at 13:43 7/23/11** (to be more precise)
or after a certain delay:
at now +5 minutes (hours, days, weeks, months, years)
Step 2: <ENTER COMMAND> ENTER
repeat step 2 as many times you need
Step 3: CTRL D to close input

atq = show a list of jobs waiting to be executed

atrm = delete a job n°<x>
ex (delete job #42) --> atrm 42

sleep = pause between commands
with ';' you can chain commands, ex: touch file; rm file
you can make a pause between commands (**m**inutes, **h**ours, **d**ays)
ex --> touch file; sleep 10; rm file <-- 10 seconds

(continued)

crontab = execute a command regularly
-e = modify the crontab
-l = view current crontab
-r = delete you crontab
In crontab the syntax is
<Minutes> <Hours> <Day of month> <Day of week (0-6, 0 = Sunday)> <COMMAND>

ex, create the file movies.txt every day at 15:47:
47 15 * * * touch /home/bob/movies.txt
* * * * * --> every minute
at 5:30 in the morning, from the 1st to 15th each month:
30 5 1-15 * *
at midnight on Mondays, Wednesdays and Thursdays:
0 0 * * 1,3,4
every two hours:
0 */2 * * *
every 10 minutes Monday to Friday:
*/10 * * * 1-5

Execute programs in the background

Add a '&' at the end of a command
ex --> cp bigMovieFile.mp4 &

nohup: ignores the HUP signal when closing the console (process will still run if the terminal is closed)
ex --> nohup cp bigMovieFile.mp4

jobs = know what is running in the background

fg = put a background process to foreground
ex: fg (process 1), f%2 (process 2) f%3, ...

Linux Bash Shell Cheat Sheet

Basic Commands

Process Management

```
w = who is logged on and what they are doing

tload = graphic representation of system load average
       (quit with CTRL C)

ps = Static process list
       -ef --> ex: ps -ef | less
       -ejH --> show process hierarchy
       -u --> process's from current user

top = Dynamic process list
While in top:
```

- q to close top
- h to show the help
- k to kill a process

```
CTRL C to top a current terminal process

kill = kill a process
       You need the PID # of the process
              ps -u <AccountName> | grep <Application>
       Then
              kill <PID> .. .. ..
kill -9 <PID> = violent kill

killall = kill multiple process's
       ex --> killall locate

extras:
       sudo halt <-- to close computer
       sudo reboot <-- to reboot
```

Create and modify user accounts

```
sudo adduser bob = root creates new user
sudo passwd <AccountName> = change a user's password
sudo deluser <AccountName> = delete an account

addgroup friends = create a new user group
delgroup friends = delete a user group

usermod -g friends <Account> = add user to a group
usermod -g bob boby = change account name
usermod -aG friends bob = add groups to a user with-
out loosing the ones he's already in
```

File Permissions

```
chown = change the owner of a file
       ex --> chown bob hello.txt
chown user:bob report.txt = changes the user owning
report.txt to 'user' and the group owning it to 'bob'
-R = recursively affect all the sub folders
       ex --> chown -R bob:bob /home/Daniel

chmod = modify user access/permission – simple way
       u = user
       g = group
       o = other

       d = directory (if element is a directory)
       l = link (if element is a file link)
       r = read (read permissions)
       w = write (write permissions)
       x = eXecute (only useful for scripts and
       programs)
```

Linux Bash Shell Cheat Sheet

Basic Commands

File Permissions (continued)

```
'+' means add a right
'-' means delete a right
'=' means affect a right

ex --> chmod g+w someFile.txt
       (add to current group the right to modify someFile.txt)

more info: man chmod
```

Flow redirection

```
Redirect results of commands:

'>' at the end of a command to redirect the result to a file
       ex --> ps -ejH > process.txt
'>>' to redirect the result to the end of a file
```

Redirect errors:

```
'2>' at the end of the command to redirect the result to a file
       ex --> cut -d , -f 1 file.csv > file 2> errors.log
'2>&1' to redirect the errors the same way as the standard output

Read progressively from the keyboard

<Command> << <wordToTerminateInput>
       ex --> sort << END <-- This can be anything you want
              > Hello
              > Alex
              > Cinema
              > Game
              > Code
              > Ubuntu
              > END
```

Flow Redirection (continued)

```
terminal output:
Alex
Cinema
Code
Game
Ubuntu

       Another example --> wc -m << END
```

Chain commands

```
'|' at the end of a command to enter another one
       ex --> du | sort -nr | less
```

Archive and compress data

```
Archive and compress data the long way:

Step 1, put all the files you want to compress in
the same folder: ex --> mv *.txt folder/

Step 2, Create the tar file:
tar -cvf my_archive.tar folder/
       -c : creates a .tar archive
       -v : tells you what is happening (verbose)
       -f : assembles the archive into one file

       Step 3.1, create gzip file (most current):
gzip my_archive.tar
       to decompress: gunzip my_archive.tar.gz

Step 3.2, or create a bzip2 file (more powerful but slow):
bzip2 my_archive.tar
       to decompress: bunzip2 my_archive.tar.bz2
```

Linux Bash Shell Cheat Sheet

Basic Commands

Archive and compress data (continued)

step 4, to decompress the .tar file:
tar -xvf archive.tar archive.tar

Archive and compress data the fast way:

gzip: tar -zcvf my_archive.tar.gz folder/
decompress: tar -zcvf my_archive.tar.gz Documents/

bzip2: tar -jcvf my_archive.tar.gz folder/
decompress: tar -jxvf archive.tar.bz2 Documents/

Show the content of .tar, .gz or .bz2 without decompressing it:

gzip:
gzip -ztf archive.tar.gz
bzip2:
bzip2 -jtf archive.tar.bz2
tar:
tar -tf archive.tar

tar extra:
tar -rvf archive.tar file.txt = add a file to the .tar

You can also directly compress a single file and view the file without decompressing:

Step 1, use gzip or bzip2 to compress the file:
gzip numbers.txt

Step 2, view the file without decompressing it:
zcat = view the entire file in the console (same as cat)
zmore = view one screen at a time the content of the file (same as more)
zless = view one line of the file at a time (same as less)

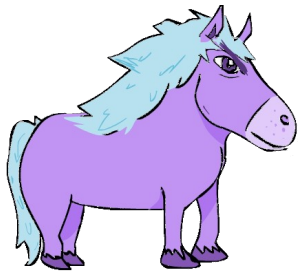
Installing software

When software is available in the repositories:
sudo apt-get install <nameOfSoftware>
ex--> sudo apt-get install aptitude

If you download it from the Internets in .gz format (or bz2) - “Compiling from source”
Step 1, create a folder to place the file:
mkdir /home/username/src <-- then cd to it

Step 2, with 'ls' verify that the file is there (if not, mv ../file.tar.gz /home/username/src/)

Step 3, decompress the file (if .zip: unzip <file>) <--
Step 4, use 'ls', you should see a new directory
Step 5, cd to the new directory
Step 6.1, use ls to verify you have an INSTALL file, then: more INSTALL
If you don't have an INSTALL file:
Step 6.2, execute **./configure** <-- creates a makefile
Step 6.2.1, run make <-- builds application binaries
Step 6.2.2 : switch to root --> su
Step 6.2.3 : make install <-- installs the software
Step 7, read the readme file



Linux Command Cheat Sheet

Share This Cheat Sheet

Basic commands

|	**Pipe (redirect) output**
sudo [command]	run < command> in superuser mode
nohup [command]	**run < command> immune to hangup signal**
man [command]	display help pages of < command>
[command] &	**run < command> and send task to background**
>> [fileA]	append to fileA, preserving existing contents
> [fileA]	**output to fileA, overwriting contents**
echo -n	display a line of text
xargs	**build command line from previous output**
1>2&	Redirect stdout to stderr
fg %N	**go to task N**
jobs	list task
ctrl-z	**suspend current task**

File permission

chmod -c -R	**chmod file read, write and executable permission**
touch -a -t	modify (or create) file timestamp
chown -c -R	**change file ownership**
chgrp -c -R	change file group permission
touch -a -t	**modify (or create) file timestamp**

Network

netstat -r -v	**print network information, routing and connections**
telnet	user interface to the TELNET protocol
tcpdump	**dump network traffic**
ssh -i	openSSH client
ping -c	**print routing packet trace to host network**

File management

find	**search for a file**
ls -a -C -h	list content of directory
rm -r -f	**remove files and directory**
locate -i	find file, using updatedb(8) database
cp -a -R -i	**copy files or directory**
du -s	disk usage
file -b -i	**identify the file type**
mv -f -i	move files or directory
grep, egrep, fgrep -i -v	**print lines matching pattern**

File compression

tar xvfz	**create or extract .tar or .tgz files**
gzip, gunzip, zcat	create, extract or view .gz files
uuencode, uudecode	**create or extract .Z files**
zip, unzip -v	create or extract .ZIP files
rpm	**create or extract .rpm files**
bzip2, bunzip2	create or extract .bz2 files
rar	**create or extract .rar files**

File Editor

ex	**basic editor**
vi	visual editor
nano	**pico clone**
view	view file only
emacs	**extensible, customizable editor**
sublime	yet another text editor
sed	**stream editor**
pico	simple editor

Directory Utilities

mkdir	**create a directory**
rmdir	remove a directory

File Utilities

tr -d	**translate or delete character**
uniq -c -u	report or omit repeated lines
split -l	**split file into pieces**
wc -w	print newline, word, and byte counts for each file
head -n	**output the first part of files**
cut -s	remove section from file
diff -q	**file compare, line by line**
join -i	join lines of two files on a common field
more, less	**view file content, one page at a time**
sort -n	sort lines in text file
comm -3	**compare two sorted files, line by line**
cat -s	concatenate files to the standard output
tail -f	**output last part of the file**

Scripting

awk, gawk	**pattern scanning**
tsh	tiny shell
" "	**anything within double quotes is unchanged except \ and $**
' '	anything within single quote is unchanged
python	**"object-oriented programming language"**
bash	GNU bourne-again SHell
ksh	**korn shell**
php	general-purpose scripting language
csh, tcsh	**C shell**
perl	Practical Extraction and Report Language
source [file]	**load any functions file into the current shell, requires the file to be executable**

Memory & Processes

free -m	**display free and used system memory**
killall	stop all process by name
sensors	**CPU temperature**
top	display current processes, real time monitoring
kill -1 -9	**send signal to process**
service [start|stop|restart]	manage or run sysV init script
ps aux	**display current processes, snapshot**
dmesg -k	display system messages

Disk Utilities

df -h, -i	**File system usage**
mkfs -t -V	create file system
resize2fs	**update a filesystem, after lvextend***
fsck -A -N	file system check & repair
pvcreate	**create physical volume**
mount -a -t	mount a filesystem
fdisk -l	**edit disk partition**
lvcreate	create a logical volume
umount -f -v	**umount a filesystem**

Misc Commands

pwd -P	**print current working directory**
bc	high precision calculator
expr	**evaluate expression**
cal	print calender
export	**assign or remove environment variable**
` [command]	backquote, execute command
date -d	**print formatted date**
$[variable]	if set, access the variable

Read the Blog Post »
bit.ly/Linux-Commands

Compiled by Alvin Khoo