Table of Contents

A cookbook for using command line tools to do everyday's job.

In everyday's job, you often encounter various types of repetitive manual tasks, such as renaming a lot of files, finding texts, processing texts and so on. These tasks take a lot of time doing manually using GUI. This book aims to leverage the power of the command line tools to make your job less tedious and more enjoyable.

This book presents "recipes" for preparing or accomplishing specific tasks. Each recipe contains a "Problem" statement and a "Solution" section. The solution section lists one or several best ways to accomplish the task that the author knows of.

Since I want to focus on having at least one workable solution that you can just copy and paste, not on teaching Bash by examples, there will not be a lot of detailed explanations about what each command does. Readers are encouraged to do extensive research about particular commands or patterns they are interested in. By doing that, they will discover even much more usage in situations they may not think of at the moment. Doing so however will require more time which normally we don't have. This is the practical reason why I chose this approach.

The recipes are ordered randomly. All commands should be applicable on most Linux distros, MacOS (there might be some missing default commands you might have to install using HomeBrew). On Windows you have to install `cygwin` or something similar. On some occasions, I use external tools from Python, Perl or Nodejs community since they're super easy to install and use, just like your default Linux or GNU commands.

This book is always a work in progress.

---

read online at:

- https://minhhh.gitbooks.io/command-line-cookbook/content/

download a .pdf, .epub, or .mobi file from:

- https://www.gitbook.com/book/minhhh/command-line-cookbook/details

contribute content, suggestions, and fixes on github:

- https://github.com/minhhh/cli-cookbook

References

- The Linux Cookbook
- Commandline fu

This chapter focus on using the shell to run and/or coordinate different programs together. Even though there are many different shell, we focus on `bash`, which is the standard on most Linux systems, including MacOS.

Running a list of commands

Problem

You want to run a list of commands in order, sometimes in parallel. Sometimes you want to run a command only if another command succeeds or fails.

Solution

To run more than one command in order, simply type each command in the order you want them to run, separating them with a semicolon `;`

```
echo 1; echo 2; echo 3;
> 1
> 2
> 3
```

To run a command only if the previous ones succeed, we can use `&&`

```
ls <file> && rm <file> -rf
```

To run a command only if the previous ones fail, we use `||`

```
ls file &> /dev/null || echo "File not exist"
> File not exist
```

To run several commands in parallel, you can run them as background process using `&` then `wait`

```
process1 &
process2 &
process3 &
process4 &
wait
process5 &
process6 &
process7 &
process8 &
wait
```

If you want to make sure that all processes succeeds together, you can use npm package `parallelshell`

```
    parallelshell "echo 1" "echo 2" "echo 3"
```

References

- http://stackoverflow.com/questions/19543139/bash-script-processing-commands-in-parallel

Converting date and time

Problem

We want to quickly convert standard date time format to POSIX time and vice versa.

Solution

The standard `date` can do this easily

Convert from date to POSIX

```
date -d "17 June 2015 15:00:00" +%s
> 1434520800

# get current date in POSIX
date +%s
```

Convert from POSIX to date

```
date -d @1232144092
> Sat Jan 17 07:14:52 JST 2009
```

By the way to set the current date

```
date --set "25 July 2014 15:00:00"
```

Simulating user inputs

Problem

Some programs require user to enter certain command before proceeding, we want to be able to send input to those programs automatically.

Solution

The `expect` program can be used for this purpose. It will detect certain text that the program output such as a question, or a prompt, then it sends the the texts that we've prepares to the program.

For instance, send ssh password automatically

```
    expect -c "ssh abc@10.0.0.10\n ;  expect password: ; send mypassword\n ; interact
"
```

Running a command for each item in a list

Problem

A super common thing you need to do is to do something repetively to a lot of files. For instance, change all the file names. For simple task like just changing the extension, you can do `mv *.png *.jpg`. But what if you want to replace spaces in the file name with `-` or `_`. Or maybe you want to add a hash to the file name. Things are a little bit more complicated.

Solution

The universal solution to all of these problems is `xargs`. Basically, you can print all of the items that you need to do something with and pass it to `xargs` to process them one by one.

The way to do this is:

```
...previous command | xargs -n 1 command
```

The `-n 1` part is to ensure that you process the list one at a time. This only work if the previous command doesn't generate string with newline and spaces. If it does then we have to specify another character as delimiter by xargs, usually the null character `\0`. With the find command we can use option -print0.

Let's try to add an extra extension to all our files in a folder

```
find . ! -path . -print0 | xargs -n 1 -0 -I {} sh -c 'mv "{}" "{}.extra"'
```

The `find . ! -path . -print0` part find all files and folder, excluding current dir `.`. Then `xargs` take each of the name as `{}`, and execute move command `sh -c 'mv "{}" "{}.extra"'` to them.

Let's see how can we replace all spaces in all filenames in a folder with `-`.

```
find . ! -path . | awk '{ str=$0; gsub(" ", "-", str); print "mv \""$0"\" \""str"\
"" "\0"; }' | xargs -n 3 -0 -I {} sh -c '{}'
```

This time we have to do a little more quoting so that we can deal with filenames with space. Basically we generate a string of the command we want to run using `awk`. Then we pass the whole command string to xargs and run it.

It's worth knowing that we can also do this directly with `awk`

```
    .... previous command | awk '{system("command \""$0"\"")}'
```

So the previous command becomes

```
    find . ! -path . | awk '{ str=$0; gsub(" ", "-", str); system("mv \""$0"\" \""str"
\"" "\0"); }'
```

In this case using `awk` is simpler. However when you have to process several parameters/lines at a time, `xargs` proves to be more straight forward.

Generating random numbers

Problem

You want to generate a random number. Or maybe you want to generate a random hash to be used as password.

Solution

Reading from `/dev/random` or `/dev/urandom` is the way to generate randomness in Linux.

To generate random number we do it like so

```
    # generate one byte of random, i.e. 0 to 255
    od -A n -t d -N 1 /dev/urandom
    > 87

    # generate 2 byte of random, i.e. 0 to 65535
    od -A n -t d -N 2 /dev/urandom
    > 30651
```

Generate random password or hash

```
    # From your current date, obviously not very random if you generate several in a r
ow. Take 32 characters only
    date +%s | sha256sum | base64 | head -c 32 ; echo

    # another way to use date
    date | md5sum

    # another way is to use the existing openssl if you have it in your system
    openssl rand -base64 32

    # get random from /dev/urandom
    cat /dev/urandom | tr -dc _A-Z-a-z-0-9 | head -c${1:-32};echo;
```

Using crontab to commit your git repo

Problem

You have a private repo that you constantly commit to. The repo is quite trivial so you don't really care about comment convention or other practices, you just want it to be updated. It's quite tedious to commit by hand.

Solution

Fortunately, we can use `cron` to automate the commit process. First, open your `crontab` like so:

```
crontab -e
```

This will open your crontab with the default editor. There might be some existing lines. What you have to do is to add a line to make auto commit to your repo, then save the text. The line you will add is:

```
*/5 * * * * (cd <path_to_your_repo> && (git add -u && git commit -m "update") || echo
"" && git pull --rebase && git push)
```

Replace `<path_to_your_repo> with the correct path to your repo. This command will make a commit with the comment "update" every 5 minutes. You can increase the commit frequency but I think 5 minutes is a good amount. You can check your current` crontab `with` crontab -l`

This chapter focus on the tools for manipulating files and directories.

Watching a directory and execute command on file change

Problem

Watch a file sets or directory and run a command when anything is added, changed or deleted.

Solution

Use python watchdog module, which has a command line tool called `watchmedo`

```
watchmedo shell-command --recursive --command 'echo ${watch_event_type}' -w -W . \
| xargs -n 1 -I {} sh -c 'if [ "{}" = "modified" ]; then clear; make unittest; fi'
```

Alternatively, can use nodejs onchange module

```
onchange 'app/**/*.js' 'test/**/*.js' -- npm test
```

Listing file and folder sizes

Problem

You want to print the sizes of all files and folders in the current folder from largest to smallest

Solution

We simply run `du` command on each file and folder in the current folder then sort them using `sort`

```
ls -A | awk '{system("du -sm \""$0"\"")}'| sort -nr | head
```

To list only folders

```
ls -Al \
| egrep '^d' \
| awk '{printf $9; for (x=10; x <= NF; x  ) {printf " "$x;}; print ""}' \
| awk '{system("du -sh \""$0"\"")}'
```

To list only files

```
ls -Al | egrep -v '^d' \
| awk '{printf $9; for (x=10; x <= NF; x  ){printf " "$x;}; print ""}' \
| awk '{system("du -sh \""$0"\"")}'
```

References

- http://groups.google.com/group/comp.unix.shell/browse_thread/thread/aebcbd0591714584/5e496ed7cfbe6eb1
- http://en.wikipedia.org/wiki/Xargs
- http://www.cyberciti.biz/faq/linux-list-just-directories-or-directory-names/

Generating random file with particular size

Problem

You want to generate a random file used for testing with a particular size.

Solution

You can use `dd` to generate file with random content like this

```
dd if=/dev/urandom of=myFile.dat bs=64M count=16
```

Printing a file in text or hex

Problem

We want to print a file with different representation. We also want to print various information related to the file.

Solution

For text file we can use various command like `cat` , `head` , `tail` , `more` , `less`

If we want to see file in hex format we can use `hexdump`

```
hexdump <file>
```

To print information about the file such as file type we can use `file` command

```
file <file>
```

Splitting and merging files

Problem

We want to split a big file into smaller files and join them back later to the original file.

Solution

Use `split` to split file easily

```
# Default split will create xaa, xab, etc files
split <FILE>
> xaa
> xab
> xac
> xad

# Split with fixed number of files, numeric suffix of 3 digits, and prefix
split <FILE> -n 10 -a 3 -d <PREFIX>

# Split with fixed file size, numeric suffix of 3 digits, and prefix
split <FILE> --bytes=1000 -a 3 -d <PREFIX>
```

To merge splitted files, simply `cat` them together

```
cat prefix* > <NEWFILENAME>
```

Converting files to different format

Problem

You want to convert a file to/from different formats

Solution

`iconv` can be used to easily convert files from one character set to another

```
# convert from UTF-8 to ISO-8859-15/latin-1
iconv -f UTF-8 -t ISO-8859-15 <infile> > <outfile>
```

`recode` can do the same thing but `in-place`

```
recode UTF8..ISO-8859-15 <infile>
```

`recode` can also be used to convert line endings

```
# convert newlines from LF to CR-LF
recode ../CR-LF <infile>

# base64 encode file
recode ../Base64 <infile>
```

`recode` can also combine transform character set, line endings and encode

```
recode utf8/Base64..l1/CR-LF/Base64 <infile>
```

Generating checksum

Problem

You want to generate different type of checksum for a file

Solution

You can use various checksum tool

```
cksum <file>
md5 <file>
shasum <file>
```

Generating mime

Problem

You want to encode and decode MIME format.

Solution

You can use `mimencode` and `mmdecode`

```
mimencode <file>
mmdecode <file>
```

Counting number of lines or characters

Problem

We want to count the number of lines, characters or words in a file

Solution

To count the number of characters, words or lines, we use `wc`

```
# this shows number of line, words, character respectively
wc <file>

# show number of lines
wc -l <file>

# show number of words
wc -w <file>

# show number of characters
wc -c <file>
```

Copying and showing progress

Problem

You constantly copying big folders or big files. You see the files are being copied, but it's quite annoying when copying big files that you don't know how much of the file has been copied.

Solution

You can use `rsync` with `-P` options

```
rsync -larP source dest
> t/a
>         19 100%    0.04kB/s    0:00:00 (xfer#843, to-check=8/1137)
> t/b
>         35 100%    0.08kB/s    0:00:00 (xfer#844, to-check=7/1137)
> ...
```

What's even better about `rsync` is that it won't copy files that are the same in the destination so the next time you change something in the source folder you can execute the same command again and it will only copy the change, not the whole folder again.

Still, when copying a folder with a lot of files, you still don't know the copying progress. In this case, we can pipe the result of `rsync` to another program called `pv`. This `pv` program will show you a progress bar and ETA time.

```
   export SOURCE=<source> DEST=<dest> && export SC=$(find "$SOURCE" | wc -l) && rsync
-vrltd  --stats --human-readable "$SOURCE" "$DEST" | pv -lep -s $SC > /dev/null
```

Even though this looks complicated, it's actually quite straight forward. First we use temporary variables to store the source and the destination folders. Then we count how many items are there in the source folder. Then we use `rsync` to copy files one by one and print them out. `pv` will pick up the number of lines and use that with the total number of items to calculate the complete percentage.

Note that if you want to copy a folder inside another folder then you should not include `/` in `<source>`, for instance, set it to `MyFolder`. If you want to copy and rename `MyFolder` to `AnotherFolder` then `<source>` should be set to `MyFolder/`.

Batch renaming

Problem

This is actually one of the most common problems that we encouter everyday. Sometime somewhere someone will name all the files incorrectly and we want to change the extensions or some part of the filename to suit our needs.

Solution

To change one kind of file name to another, we use `bash` for loop with `find`

```
# change from .html to .txt
for file in *.html; do
    mv "$file" "`basename $file .html`.txt"
done
```

You can probably do the same with `awk` as well. The principle is to list the file first then rename them one by one.

However, there's a more convenient way to batch rename, which is the `rename` command. You can use it like so

```
rename 's/\.html$/\.txt/' *.html
```

The syntax is `rename "regex-rule" <files>`. So if you can do some simple regex, it's better to do this way.

Another example is to change all filenames to lower case or upper case. Again, you can use `rename`

```
# uppper to lower
rename -f 'y/A-Z/a-z/' *

#  lower to uppper
rename -f 'y/a-z/A-Z/' *
```

Counting number of lines or characters

Problem

We want to count word frequency and sort it from top to bottom.

Solution

To make a list of word frequency in a document, we can combine `wc`, `sort` and `awk` like so

```
cat ~/bitbucket/wiki/about.md | tr ' ' '
' | sort | uniq -c | sort -rnk1

# result
51 to
39 and
36 I
31 of
31 a
30 the
...
```

Comparing 2 folders

Problem

You want to compare the content of 2 folders recursively.

Solution

You can use `diff` with `-rq` option

```
diff -rq <folder1> <folder2>
```

If you want more detailed report of the differences, you can use a `Nodejs` tool called `dir-compare`. First, install nodejs, then install `dir-compare` by this command `npm install dir-compare -g`. Then you can use it like this:

```
dircompare -ca <folder1> <folder2>
```

There are other options such as to exclude directories or files by name. You can discover more by using `dircompare -h`

This chapter focuses on text manipulation.

Searching text from files

Problem

You want to search for text in a lot of files swiftly.

Solution

You can use `grep` or `egrep`

```
#list only file name
find . | xargs grep 'string' -sl
find / -type f -print0 | xargs -0 grep -l "test"

# print text and file name
grep -r "redeem reward" /home/tom

# egrep with regular expression
egrep "^\s+\"" file1

# grep excluding files
grep -ircl --exclude=*.{png,jpg} "foo=" *
grep -Ir --exclude="*\.svn*" "pattern" *
```

However the much better solution is to use `ag` or `ack`

```
ag -Q --smart-case --ignore=pack*.js --ignore=Code/tag \
--ignore-dir=build --ignore-dir=Code/JSON --ignore-dir=Tools --js "test"

ack -Q --smart-case "test" --js --ignore-file=match:/packed.*\.js/ \
--ignore-file=is:Code/tag --ignore-dir=build --js "test"
```

Removing duplicated lines

Problem

You want to remove duplicated lines in a file or from stdin.

Solution

You can combine `uniq` and `sort` to achieve this.

```
sort garbage.txt | uniq -u
cat garbage.txt | sort | uniq -u
```

Printing a range of lines

Problem

You want to print a range of lines from a file or from stdin, not the whole thing. For instance, you may want to print only the first 3 lines, or the last 5 lines, or everything except the first line, or everything except the last 2 lines.

Solution

First, we can count the number of lines in a file like this

```
wc -l <file>
cat <file> | wc -l
```

Print the first `n` line with `head`

```
head -n 10 <file>
```

Print last `n` line with `tail`

```
tail -n 10 <file>
```

Print everything except the first `n` line with `tail`

```
tail -n +7 <file>
```

Print everything except the last `n` line with `head`

```
head -n -2 <file>
```

Print from line `x` to line `y` with `sed`

```
sed -n "1,3p" <file>
```

Converting tab to space

Problem

You want to convert tab to space and vice versa.

Solution

To convert from tab to space you can use `expand`

```
    # convert tab to 4 space in all java files
    find . -name '*.java' ! -type d -exec bash -c 'expand -t 4 "$0" > /tmp/e && mv /tm
p/e "$0"' {} \;
```

To convert all 4 spaces to tab, use `unexpand`

```
    unexpand -t 4 <input_file>
```

Comparing 2 text files

Problem

You want to compare 2 text files side by side.

Solution

Linux already has a tool to do this called `diff`

```
    diff file1 file2
```

The output will be something like this

```
1c1
< 1
---
> 2
```

where the `<` part is in the first file only and the `>` part is in the second file only.

If you want more visual diff you can use `colordiff`

Sorting lines based on a certain field

Problem

You want to sort a list of lines from a file or from stdin based on a certain field, provided all the lines follow the same format.

Solution

First, you can sort the whole line with `sort`.

For instance, you can sort lines in `/etc/password`, which will sort by the username since the username is the first field in each line.

```
    # sort password file by username
    sort /etc/passwd

    # original content
    _kadmin_changepw:*:219:-2:Kerberos Change Password Service:/var/empty:/usr/bin/fal
se
    _devicemgr:*:220:220:Device Management Server:/var/empty:/usr/bin/false
    _webauthserver:*:221:221:Web Auth Server:/var/empty:/usr/bin/false
    _netbios:*:222:222:NetBIOS:/var/empty:/usr/bin/false
    _warmd:*:224:224:Warm Daemon:/var/empty:/usr/bin/false
    _dovenull:*:227:227:Dovecot Authentication:/var/empty:/usr/bin/false

    # content after sorting
    _devicemgr:*:220:220:Device Management Server:/var/empty:/usr/bin/false
    _dovenull:*:227:227:Dovecot Authentication:/var/empty:/usr/bin/false
    _kadmin_changepw:*:219:-2:Kerberos Change Password Service:/var/empty:/usr/bin/fal
se
    _netbios:*:222:222:NetBIOS:/var/empty:/usr/bin/false
    _warmd:*:224:224:Warm Daemon:/var/empty:/usr/bin/false
    _webauthserver:*:221:221:Web Auth Server:/var/empty:/usr/bin/false
```

However, most of the time we want to sort the file based on a field in the middle. In this case we use sort by `field` feature.

```
# Sort by the second field
cat somefile.txt | sort -rnk2

# original content
x   1   2
x   2   2
x   3   2
x   12   2
x   9   2
x   3   2

# content after sorting
x   12   2
x   9   2
x   3   2
x   3   2
x   2   2
x   1   2
```

In a more general case, we want to sort by calculating a value based on some fields, for instance, the ratio between field 2 and field 3. In such cases, we will use `awk` to calculate the derived field then use `sort` on the final result

```
    # sort based on field 3 / field 2 then print the result at the beginning of the li
ne
    cat somefile.txt | awk '{ratio = $2/$1; print ratio, $0;}' | sort -rnk1

    # original content
    x   1   2
    x   2   2
    x   3   2
    x   12   2
    x   9   2
    x   3   2

    # content with the calculated value inserted as the first field: cat somefile.txt
| awk '{ratio = $2/$1; print ratio, $0;}'
    2 x   1   2
    1 x   2   2
    0.666667 x   3   2
    0.166667 x   12   2
    0.222222 x   9   2
    0.666667 x   3   2

    # content after sorting
    2 x   1   2
    1 x   2   2
    0.666667 x   3   2
    0.666667 x   3   2
    0.222222 x   9   2
    0.166667 x   12   2
```

Real world example: Counting unique ip access in apache log in a month

```
    grep Jan/2004 access.log | grep foo.php | \
    awk '{ print $1; }' |  sort -n | uniq -c | \
    sort -rn | head
```

Merge content of 2 files side by side

Problem

You want to show the content of 2 files next to each other, line by line. For instance, one file contains id and the other contains names.

Solution

You can use `paste` to achieve this.

Suppose `file1` contains names

```
Mark Smith
Bobby Brown
Sue Miller
Jenny Igotit
```

and `file2` contains numbers

```
555-1234
555-9876
555-6743
867-5309
```

You can use `paste` like so

```
    # Merge with default separator: tab
    paste file1 file2
    > Mark Smith    555-1234
    > Bobby Brown    555-9876
    > Sue Miller    555-6743
    > Jenny Igotit    867-5309

    # merge with delimiter comma (,)
    paste -d, file1 file2
    > Mark Smith,555-1234
    > Bobby Brown,555-9876
    > Sue Miller,555-6743
    > Jenny Igotit,867-5309
```

Joining all lines in a file

Problem

You want to join all lines in a file, often separated by a comma

Solution

You can use `paste` to achieve this like so

```
# Join with tab separate each line
paste -s <filename>

# join with delimiter comma
paste -d, -s <filename>
```

Adding text to the beginning of a file

Problem

You want to quickly add a piece of text to the beginning of a file.

Solution

The simplest way is to just print the text together with the content of the file to a temporary file, then copy the temporary file to the original file.

```
echo 'Begin' | cat - <file> > temp && mv temp <file>
```

Another way is to use `sed` program to insert the text to the beginning of the file and use edit in place functionality of `sed` so that we don't have to create temporary file.

```
sed -i '1s/^/Begin\n/' <file>

# A shorter version
sed -i '1iBegin' <file>
```

Replacing strings

Problem

You want to replace one string by another in one or many files.

Solution

You can use `sed` to do various string manipulation tasks

To replace `old-word` by `new-word`

```
sed -i 's/old-word/new-word/g' *.txt

#in mac
sed 's/old-word/new-word/g' -i '' *.txt
sed 's/old-word/new-word/g' *.txt
```

Some times the words contain single quotes, in these cases, we have to escape them like so

```
sed 's/old-word/new'\''word/g' *.txt
```

If you want `sed` to read from standard input, you have to use `-e` option

```
echo "hello world" | sed -e 's/hello/hi/g'
> hi world
```

Changing cases

Problem

You want to convert lower case to upper case and vice versa

Solution

You can use multiple tools to do this task. One simple solution is the `tr` program

```
# convert upper to lower case
echo "HELLO" | tr '[:upper:]' '[:lower:]'
> hello

# convert lower case to upper case
echo "hello" | tr '[:lower:]' '[:upper:]'
> HELLO
```

Deleting lines that contain a specific string

Problem

In a lot of occasions, you would want to remove a particular line in a file if it exists.

Solution

You can use a lot of tools to do this such as `awk`, `bash`, `sed`, etc.

Using `sed` is pretty intuitive. To remove a line contains a specific string we do like this

```
# Will remove all lines containing helloworld
sed -i '/helloworld/d' ./infile
```

Sometimes we want to match the whole line, we can do like this

```
# Will remove all lines which are exactly `helloworld`
sed -i '/^helloworld$/d' ./infile
```

This chapter focuses on finding things in the file system.

Finding files based on name

Problem

You want to find files or folders in a directory by name.

Solution

Use `find` to find files or folders in a folder by name.

Find all files or folder in the current directory whose name is `password.txt`.

```
    # This command would match any files whose name consisted of the letters `password
.txt', regardless of case, including 'password.txt', 'PASSWORD.TXT', and 'password.TXT
'.
    find . -iname password.txt
```

We can use some wildcard patterns to make the searching easier in case we don't know the exact names, or we just want to find all files with similar patterns.

```
    # search all files begin with `pass`
    find . -iname 'pass*'

    # file all files with a certain extension
    find . -iname '*.png'
    find . -iname '*.txt'

    # search all files whose name contains `pass` somewhere
    find . -iname '*pass*'
```

To find only file or folder we have to specify the type like so

```
    # search only files
    find . -type f -iname 'win*'

    # search only folder
    find . -type d -iname 'win*'
```

To exclude specific folders we use the `-not -iname` option

```
find -name "*.js" -not -iname "hello.js"
```

To exclude specific folders we use the `-not -path` option

```
find -name "*.js" -not -path "./directory/*"
```

Running commands on the files you find.

```
find . -name '*.md' -exec echo 'Found {}' ';'
> Found ./android_cli.md
> Found ./awk.md
> Found ./bash.md
> Found ./curl.md
> Found ./daemons.md
> Found ./docker.md
> ...
```

This is similar to "Running a command for each item in a list" part. You can use `awk` or `xargs` to do more advanced things. For simple operations you can use `find`.

Finding files based on size

Problem

You want to find the largest file or folders, maybe recursively. You also want to find files that are bigger/smaller than X bytes

Solution

Find the largest file/folder non-recursively OR sort files and folders by size

```
ls -A | awk '{system("du -sh \""$0"\"")}'| sort -hr | head
```

Find the largest file in a folder and all subfolders recursively

```
find . -type f -print0 | xargs -0 -n 1 du -sh | sort -hr | head

# display in block of 1024-byte
find . -type f -print0 | xargs -0 -n 1 du -sk | sort -nr | head
```

This command use `find` to search for all file recursively. The option `-print0` removes the need for `sed` to escape spaces since all fields now are separated by null character. `args -0` makes sure we use null separator.

To find files smaller/larger than X bytes we will also use `find` command with the `-size` option

```
// find files larger than 4096 bytes
find . -type f -size +4096c

// find files smaller than 1M
find . -type f -size -1M
```

Options for the `-size` switch

```
    -size n[ckMGTP]
        True if the file's size, rounded up, in 512-byte blocks is n.  If n is followe
d by a c, then the primary is true if the
        file's size is n bytes (characters).  Similarly if n is followed by a scale in
dicator then the file's size is compared to n
        scaled as:

        k       kilobytes (1024 bytes)
        M       megabytes (1024 kilobytes)
        G       gigabytes (1024 megabytes)
        T       terabytes (1024 gigabytes)
        P       petabytes (1024 terabytes)
```

Finding files in a folder

Problem

You want to find a file by name in a folder, but also want to exclude certain folder in the find path. Also, you want to run some command after finding the files.

Solution

The `find` tool can find file/folders by name, exclude certain folders/files and execute command on found items.

Find files by name, excluding the git folders

```
find . -not -path "./.git/*" -iname "my.png"
```

Remove all .svn files in a directory

```
find . -name ".svn" -exec rm -rf {} \;
```

Counting the number of files or folders

Problem

You want to count the number of files and/or folders in a certain folder

Solution

The simplest solution is to combine `find` and `wc`

To count all files and folders recursively

```
# Count all files and folders in the current folder
find . | wc -l
> 5254

# Count all files and folders in a certain folder with name <name>
find <name> | wc -l
> 5054
```

To count only files

```
find . -type f | wc -l
> 4232
```

To count only folders (this will also count the current folder so remember to subtract 1 if you want the correct number)

```
find . -type d | wc -l
> 432
```

To count folders and files in the current folder only, not recursively, use `-depth 1` parameter

```
find . -type d -depth 1 | wc -l
> 29

find . -type f -depth 1 | wc -l
> 2
```

This chapter introduces several simple admistratrative tasks. Most of these commands should be run using `root` account.

Shutting down

Problem

You want to shutdown system, sometimes immediately, sometimes at a certain time or after a certain duration.

Solution

Use the `shutdown` command with `root` privilege.

To immediately shut down and halt the system

```
sudo shutdown -h now
```

To immediately reboot the system

```
sudo shutdown -r now
```

You can optionally send a warning message to all user with `-c` option

```
sudo shutdown -h now "The system is being shut down now!"
```

To shut down the system at a certain time

```
# At 4.23 AM
sudo shutdown -h 4:23

# At 8.00 PM
sudo shutdown -h 20:00
```

To shut down and halt the system after a period of time

```
# In 5 minutes
sudo shutdown -h +5
```

To cancel a shutdown

```
sudo shutdown -c
```

Execute a command at a specified time

Problem

You want to run a command at a specified time, or at a certain amount of time from now.

Solution

Use `at` command to run arbitrary commands and scripts at a specified time.

There are 2 ways to run `at` : from user input or from a script

To run from user input

```
at <time>
...type in the commands
Ctrl-D
```

To run from a script

```
at <time> -f <script>
```

To run a command immediately

```
at now
```

To run a command at a specific time in the future

```
# run this job at 10:00 Jan 10, 2015
at 10:00 Jan 10 2015

# run this job at midnight, noon, or teatime, respectively
at midnight
at noon
at teatime

# run this job at noon today or tomorrow
at noon today
at noon tomorrow
```

To run a command after an amount of time has elapsed from now, just add `+` to the time

```
# run this job after 3 minutes
at now + 3 minutes

# run this job at 4pm 3 days from now
at 4pm + 3 days
```

To list current jobs, use `atq`

```
atq
```

To remove a job listed by `atq`, use `atrm`

```
atrm 10
```

`batch` is similar to `at`, but it only executes command when system load levels permit, i.e., when the load average drops below 1.5.

Pausing execution

Problem

You want to pause/sleep for some seconds on the command line.

Solution

You can use `sleep` and `usleep`

```
# sleep for 1 second
sleep 1

# sleep for 1000 milliseconds
usleep 1000
```

Stuff that does not fit anywhere should go here

Doing mathematics

Problem

We want to do simple mathematics

Solution

Use `bc` command

```
echo "5.1 * 2" | bc - l
> 10.2

echo "scale=10; 1/2" | bc -l
> .5000000000
```