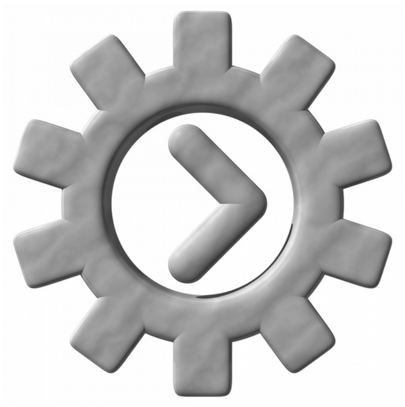
100 Command Line Tools

For Windows, Linux and Mac OS/X



How to do things fast,
with the same commands,
on every computer!

Contents

SFK Command Reference

Index

Introduction

How to do things the same way on all computers

Quick file exchange between machines, find duplicate files, find and replace text, list directory tree sizes, and many other functions for daily tasks: normally this requires hour-long installations and configurations of masses of separate programs, often interrupted by missing or wrong versioned libraries ("do you have the latest .NET/Java/Cygwin/Qt?"). There may be a thousand tools for the same task, but you may have to install 5 of them, just to find out they work different than expected, until the 6th may work, causing a spammed registry and time waste in general.

Therefore the **Swiss File Knife** - aka **SFK** - was developed. It is a small, single binary for the command line that runs instantly, without installation.
It contains 100 tools for the most needed tasks, with the same basic syntax for Windows, Linux and Mac OS/X.

How to get the Swiss File Knife up and running anywhere.

Download the executables for Windows, Linux or Mac OS/X

By web browser: go to

`http://stahlworks.com/sfk/`

then click on one of the top links to download your binary instantly.
Alternatively, look on SourceForge:

`http://sourceforge.net/projects/swissfileknife/`

or on a Linux text console, use one of these:

`wget http://stahlworks.com/sfkux`	for current 32 bit systems
`wget http://stahlworks.com/sfkux64`	for current 64 bit systems
`wget http://stahlworks.com/sfkuxold`	for older 32 bit systems (like DSL, using lib5)
`wget http://stahlworks.com/sfkarm`	for 32 bit ARM systems

If your system has no wget command then try curl instead, like:

```
curl -o sfk http://stahlworks.com/sfkux
```

The Apple Mac OS/X binaries are available by:

`curl -o sfk http://stahlworks.com/sfkmac`	for current Intel based Macs
`curl -o sfk http://stahlworks.com/sfkmacold`	for PowerPC based Macs

Self compile: on systems for which no binary is available you may download the sourcecode from the SourceForge link (.zip or .tar.gz). Make sure the g++ or gcc compiler is installed on your system. Then type:

```
g++ sfk.cpp sfkext.cpp sfkpack.cpp -o sfk
```

Transfer of SFK without internet access:

If the target machine has any connection to a local network, try the following:

SFK Instant HTTP Server for easy file exchange

on another machine where you have SFK already, type

```
sfk httpserv -port=9090
```

and make sure the sfk binary is located in the current folder.
on the target machine, open a web browser and access:

```
http://othermachine:9090/
```

or on a Linux/Mac console, type one of:

```
wget http://othermachine:9090/sfk
curl -o sfk http://othermachine:9090/sfk
```

further reading:
httpserv tutorial on page 42,
reference on page 260.

If that fails (no browser, no gui, no wget or curl command), check if there is an "ftp" command on the target. If so, try:

SFK Instant FTP Server for easy file exchange

on a machine where you have SFK already, type:

```
sfk ftpserv
```

it will tell you the machine's IP address. then, on the target machine, type:

```
ftp ipaddress
```

and if the login succeeds, try:

```
bin
get sfk.exe
```

If ftp cannot connect to the server then try to run ftpserv as administrator. If get fails, check if the ftp client on the target accepts the command:

```
passive
```

then try to "get" again (ftp creates a new connection per file download, which is often blocked by firewalls. the passive command changes the way in which those connections are created.)

further reading:
ftpserv tutorial on page 42,
or the reference on page 261.

How to prepare the SFK binary under Linux:

After download, you have to type

```
mv sfkux sfk
chmod +x sfk
```

to enable execution (the 'x' flag) of sfk. Then simply type

```
./sfk
```

to get it running (the "./" is often needed as the PATH may not contain the current directory ".").

Where to place the SFK executable:

Recommendation for Windows

Create a directory structure

```
c:\app\bin
```

then copy sfk.exe to c:\app\bin. Then extend the Windows Shell Path like

```
set PATH=%PATH%;c:\app\bin
```

which is best done in a batch file like `c:\app\init.bat`, so after opening CMD.EXE just type `c:\app\init` to extend the path. Also make sure your Windows Shell (CMD.EXE) supports command auto-completion and copy/paste of text (the QuickEdit and Insert setting), otherwise it is very hard to use!

further reading:
Windows CMD.EXE configuration,
sfk help shell on page 371.

If you create a collection of batch files (e.g. through the "sfk alias" command on page 306) it is most convenient to store them in c:\app\bin as well, as this path is short and contains no blank characters. Further tools can be installed parallel to "bin" into c:\app.

Recommendation for Linux and Mac OS/X

Type "cd" then "pwd" to find out what your account's home directory is. Within your home directory (e.g. /home/users/youruserid/) create a directory "tools" by

```
mkdir tools
```

then rename sfk-linux.exe to sfk, and copy that into the tools dir.
Extend the PATH like:

```
export PATH=$PATH:/home/users/youruserid/tools
```

then you should be able to run sfk by typing "sfk".

By default, there are no colors, as it is not possible to autodetect the background color under Linux/Mac. If you like colorful output then read on under "sfk help color" on page 373.

The different editions of SFK: OSE, Base, XE

Three different types of SFK binaries exist:

- **SFK OSE** - the Open Source Edition: this is what you get when compiling the available source codes. It contains 90 percent of SFK functionality but not the commands xfind, xhexfind, xex, extract as well as the commercial commands xed and xreplace.
- **SFK Base+XD:** these are the binaries you can download from stahlworks.com and SourceForge. They also contain the closed source functions xfind, xhexfind, xex and extract, as well as the help text for xed, and a demo of xreplace that cannot write files.
- **SFK XE or Extended Edition:** this is the commercial edition of SFK available from stahlworks.com. It contains a high performance implementation of sfk replace, the sfk xed command and the sfk xreplace command. Furthermore it can search .zip, .tar.gz and .tar.bz2 file contents directly with several commands. SFK Base+XD contains a demo of this, allowing to read the first 1000 bytes of every .zip file entry.

Some Linux distributions allow installation of SFK via their package managers, however these will be SFK OSE binaries. If in any doubt, type

```
sfk ver -own
```

and it will tell it's own edition, like:

```
Base/XD windows-any     1.9.3
```

SFK Tutorial

A step by step introduction into the most
popular commands of Swiss File Knife.

List all files of a folder, and all sub folders

Everyone knows that `dir mydir` on Windows, or `ls mydir` on Linux/Mac shows the filenames in the top level of a folder mydir, without it's sub folders.

If, however, you want to list all files in mydir and all it's sub folders,
as a flat list of filenames with full path each, then use

```
   sfk dir mydir
```

example output:

```
   mydir\project1\01-make-all.sh
   mydir\project1\app\gui\base\Tools.cpp
   mydir\project1\app\gui\base\Tools.hpp
   mydir\project1\app\gui\login\Screen.cpp
   mydir\project1\config.h
   mydir\project1\config.h.bak
   mydir\project1\save\config.h
   mydir\project1\save2\config.h
   mydir\project1\save3\config.h
   mydir\project1\tmp\trash1.txt
   mydir\project1\tmp\trash2.txt
   mydir\project1\tmp\trash3.txt
   mydir\project1\tools\include\Tools.hpp
   mydir\project1\tools\include\Tools.hpp.bak
   mydir\project1\tools\new.myscm\sub1.txt
   mydir\project1\tools\org.myscm\sub1.txt
   mydir\project1\tools\source\.myscm\sub3.txt
   mydir\project1\tools\source\other1.myscm
   mydir\project1\tools\source\other1.myscm.bak
   mydir\project1\tools\source\save\.myscm
   mydir\project1\tools\source\save\.myscm-file.txt
   mydir\project1\tools\source\save\Tools.cpp
   mydir\project1\tools\source\Tools.cpp
   mydir\project1\tools\source\Tools.tmp
   25 files, 18 dirs, 2828 bytes.
```

Notice that sub folder traveling is **default** with most SFK commands, so you don't have to use an extra option for that. This is because, if I want to do something "with all files of a folder", in most cases I literally mean **all** files.

Instead of "sfk dir" you may also use "sfk list" which produces just the list of filenames, without the "files, dirs, bytes" info.

List only selected files in selected sub folders

In the above example, we notice two kinds of files:
- live files we are actively working with
- backup or trash files and folders named tmp, bak, save.

In most cases, we want to
- list all files of that folder
- except for files within folders having tmp or save in their name
- and except for files ending with .bak or .tmp.

This can be done with SFK by:

```
sfk dir -dir mydir -subdir !tmp !save -file !.bak !.tmp
```

example output:

```
mydir\project1\01-make-all.sh
mydir\project1\app\gui\base\Tools.cpp
mydir\project1\app\gui\base\Tools.hpp
mydir\project1\app\gui\login\Screen.cpp
mydir\project1\config.h
mydir\project1\tools\include\.myscm\sub2.txt
mydir\project1\tools\include\Tools.hpp
mydir\project1\tools\new.myscm\sub1.txt
mydir\project1\tools\org.myscm\sub1.txt
mydir\project1\tools\source\.myscm\sub3.txt
mydir\project1\tools\source\other1.myscm
mydir\project1\tools\source\Tools.cpp
12 files, 13 dirs, 1376 bytes.
```

Wildcards are default and need not to be specified in most cases. This means that `!save` actually means `!*save*` - i.e. excluding every sub directory that has save somewhere in it's name, like save, save2, save3 etc.

Under Linux/Mac you have to use a colon ":" instead of "!" because the command shell misinterprets "!" as some command for itself.
So use instead:

```
sfk dir -dir mydir -subdir :tmp :save -file :.bak :.tmp
```

Listing files using wildcards

To list files within sub folder names containing words "new" and "scm" use

```
   sfk list -dir mydir -subdir new*scm
```

example output:

```
   mydir\project1\tools\new.myscm\sub1.txt
```

Under Linux/Mac you must surround anything with * or ? by double quotes because the command shell misinterprets "*" as some command for itself. Alternatively you may use % as a replacement for "*". So use one of:

```
   sfk list -dir mydir -subdir "new*scm"
   sfk list -dir mydir -subdir new%scm
```

for all Linux/Mac syntax details see page 87.

List the latest or biggest files

Which files were changed most recently within mydir? Find out by:

```
   sfk list -late mydir
```

example output:

```
   2015-01-18 06:47:54 mydir\project1\app\gui\base\Tools.cpp
   2015-01-18 13:44:17 mydir\project1\tools\source\save\.myscm
   2015-02-28 08:54:20 mydir\project1\tools\source\other1.myscm
   2015-02-28 08:54:20 mydir\project1\tools\source\Tools.cpp
   2015-02-28 08:54:20 mydir\project1\tools\source\Tools.tmp
```

And what are the biggest files in mydir?

```
   sfk list -big mydir
```

example output:

```
     41 mydir\project1\save2\config.h
     56 mydir\project1\save\config.h
    171 mydir\project1\config.h
   1074 mydir\project1\tools\source\Tools.cpp
   1210 mydir\project1\tools\source\Tools.tmp
```

further reading:
sfk list, *the full syntax reference on page* 88.

Find a filename quickly in the current directory tree

You are standing within a folder and know that a file having `foo` somewhere in it's path- and/or filename exists. But you don't know exactly where. This can be solved by

```
sfk filefind foo
```

example output:

```
project1\tools\source\BarFoo.cpp
```

So, there is a file "BarFoo" in a sub folder `project1\tools\source` .
Notice that **case insensitive search is default** with every SFK command, therefore "foo" finds both "foo" and "Foo". Because this quick local filename search is needed so often, you may also type:

```
sfk :foo
```

Which does the same as "filefind foo".
Another example:

```
sfk :tool*sub2
```

may find:

```
project1\tools\include\.myscm\sub2.txt
```

as this contains "tool" in it's **path** and "sub2" in it's **filename**.

Under Linux/Mac use instead:

```
sfk :tool%sub2
```

as otherwise a * wildcard would be misinterpreted by the shell
and not given to SFK.

full syntax and examples
in the reference on page 93.

List different files between two folders

I have files in a folder "step1". I make a copy of the whole folder as "step2" and continue working within "step2". After some hours I wonder which files are different compared to the old folder.

```
sfk list -sincedir step1 step2
```

tells:

```
[dif] step2\base.php
[dif] step2\classes\tree.class.php
[dif] step2\index.php
[add] step2\organizer.php
[add] step2\tasks.php
```

meaning:

- 3 files that exist in both folders are different
- 2 files have been created in step2 that did not exist in step1

Note that files which were deleted in folder step2 are not shown. These can be found by running a reverse folder comparison:

```
sfk list -sinceadd step2 step1
```

tells:

```
[add] step1\queuescanner.php
```

so the file queuescanner.php was deleted in step2.

find all sfk list options
on page 88.

Run a command on all files of a folder

I want to collect all .jpg files in a folder mydir like

```
mydir\Formats\06-binary.jpg
mydir\myproj\app\gui\base\GreenFoo.jpg
mydir\myproj\app\gui\login\Door.jpg
mydir\myproj\tools\BackButton.jpg
mydir\myproj\tools\Home.jpg
```

into a single flat folder called "overview". This can be done by:

```
sfk list mydir .jpg +run "copy $qfile overview"
```

on Linux/Mac: use #qfile instead of $qfile.
example output:

```
[simulating:]
copy "mydir\Formats\06-binary.jpg" overview
```

```
   copy "mydir\myproj\app\gui\base\GreenFoo.jpg" overview
   copy "mydir\myproj\app\gui\login\Door.jpg" overview
   copy "mydir\myproj\tools\BackButton.jpg" overview
   copy "mydir\myproj\tools\Home.jpg" overview
   [add -yes to execute.]
```

remarks:

- first list the files by "sfk list".
- then add "+run" as a chained command.
- qfile means "filename enclosed in double quotes",
  in case that any filename contains blank characters.
- finally add "-yes" to really run the commands.

*full sfk run syntax
on page 201.*

Run a command on files that differ between two folders

Back to the example where I listed all files different between two folders named "step1" and "step2". If I want to run a difference viewer tool like WinMerge on the different files this can be done by:

```
sfk list -sincediff step1 step2 +run "winmerge $qsince $qfile"
```

on Linux/Mac: use # instead of $.
remarks:

- now we don't use -sincedir but -sincediff so it will select only *files that exist in both folders and which are different,* but not *added* files.
- then this is passed to +run
- $qsince references the filename from step1, $qfile the one from step2, and again "q" means to enclose all filenames in double quotes
- this assumes that a tool winmerge.exe exists in a folder listed in the PATH environment variable .

the simulation output tells exactly what would be done:

```
   [simulating:]
   winmerge "step1\base.php" "step2\base.php"
   winmerge "step1\classes\tree.class.php"
            "step2\classes\tree.class.php"
   winmerge "step1\index.php" "step2\index.php"
   [add -yes to execute.]
```

finally add -yes to run the commands. To stop inbetween press CTRL+C.

*full sfk run syntax
on page 201.*