

touch

Touch is usually a program, but it can be a shell built-in such as with `busybox`.

The `touch` program by default changes the access and modification times for the files listed as arguments. If the file does not exist, it is created as a zero length file (unless you use the `-c` option.)

You can also set either or both of the times to arbitrary values, such as with the `-t`, `-d`, `-B`, and `-F` options.



Backquotes and textual substitution

If you surround a command with backquotes, the standard output of the command is substituted for the quoted material.

For instance,

```
[langley@sophie 2006-Fall]$ echo `ls 0*tex`  
01-introduction.tex 02-processes.tex 03-shells1.tex 03-shells2.tex 04-shells3.tex  
[langley@sophie 2006-Fall]$ echo `egrep -l Langley *`  
03-shells2.tex Syllabus-Fall.html Syllabus-Fall.html.1 Syllabus Summer.html  
[langley@sophie 2006-Fall]$ now=`date`  
[langley@sophie 2006-Fall]$ echo $now  
Mon Sep 11 09:55:09 EDT 2006
```



Backquotes and textual substitution

```
if [ `wc -l < /etc/hosts` -lt 10 ]; then echo "lt"; fi  
# use '`<`' to prevent filename from
```



xargs

```
xargs COMMAND -n N [INITIAL-ARGUMENTS]
```

`xargs` reads from `stdin` to obtain arguments for the `COMMAND`. You may specify initial arguments with the `COMMAND`. If you specify `-n N`, then only up to `N` arguments are given to any invocation of `COMMAND`. For instance,

```
[langley@sophie 2006-Fall]$ cat /etc/hosts | xargs -n 1 ping -c 1  
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.075 ms
```



```
--- 127.0.0.1 ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.075/0.075/0.075/0.000 ms, pipe 2
```

```
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
```

```
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=0 ttl=64 time=0.060 ms
```

```
--- localhost.localdomain ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.060/0.060/0.060/0.000 ms, pipe 2
```

```
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
```

```
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=0 ttl=64 time=0.071 ms
```



The for statement

```
for name in LIST0 ; do LIST1 ; done
for name ; do LIST1 ; done    # useful for scripts
for (( EXPR1 ; EXPR2 ; EXPR3 )) ; do LIST1 ; done
```

In the last form, `EXPR?` are evaluated as arithmetic expressions.



The for statement

```
[langley@sophie 2006-Fall]$ for (( ip = 0 ; ip < 5 ; ip = ip+1)) do echo $ip ; done
0
1
2
3
4
```



The for statement

```
for i in `cat /etc/hosts`  
do  
    ping -c 1 $i  
done
```



break and continue statements

`break` terminates the current loop immediately and goes on to the next statement after the loop. `continue` starts the next iteration of a loop.



break and continue statements

For example,

```
for name in *
do
  if [ -f "$name" ]
  then
    echo "skipping $name"
    continue
  else
    echo "process $name"
  fi
done
```



expr

You can use `expr` to evaluate arithmetic statements, some regular expression matching, and some string manipulation. (You can also use either `bc` or `dc` for more complex arithmetic expressions.)



expr

```
files=10
dirs='expr $files + 5'
limit=15
if [ 'expr $files + $dirs' < $limit ]
then
  echo 'okay'
else
  echo 'too many!'
fi
```



awk

One of the more powerful programs found on Unix machines is `awk`, and its updated versions, `nawk` and `gawk`.

It is most useful for handling text information that is separated into a series of uniform records. The most common one that it handles is records of one line, divided by either column numbers or by a field separator. For instance, handling the password file is a snap with `awk`.



awk

The password file on a Unix machine looks something like:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
```



awk

You can quickly get a list of usernames into a single string variable with:

```
[langley@sophie 2006-Fall]$ usernames='awk -F: '{print $1}' /etc/passwd'
[langley@sophie 2006-Fall]$ echo $usernames
root bin daemon adm lp sync shutdown halt mail
[langley@sophie 2006-Fall]$ usernames='awk '{print $1}' FS=: /etc/passwd'
[langley@sophie 2006-Fall]$ echo $usernames
root bin daemon adm lp sync shutdown halt mail
```



awk

Fundamentally, `awk` scripts consist of a series of pairs:

```
PATTERN { ACTION }
```



awk

where the PATTERN can be a

☞ */regular expression/*

☞ *relational expression*

☞ *pattern-matching expression*

☞ *BEGIN* or *END*



awk

By default, the record separator is a newline so `awk` works on a line-by-line basis by default.

If no `PATTERN` is specified, then the `ACTION` is always taken for each record.

If no `ACTION` specified, then the each records that matches a pattern is written to `stdout`.



awk

You can specify that an ACTION can take place before any records are read with the keyword BEGIN for the PATTERN.

You can specify that an ACTION can take place after all records are read with the keyword END for the PATTERN.

With PATTERNS, you can also negate (with !) them, logically “and” two PATTERNS (with &&), and logically “or” two PATTERNS (with ||).



awk

Some examples of regular expressions in awk:

```
[langley@sophie 2006-Fall]$ awk '/[Ll]angley/ {print $0}' /etc/passwd
langley:x:500:500:Randolph Langley:/home/langley:/bin/bash
[langley@sophie 2006-Fall]$ awk '/^#/ ' /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
```



awk

- ☞ `$0` refers to the whole record, `$N` refers to the Nth field in a record
- ☞ `NF` refers to the number of fields in a record (example, `awk -F: 'END print NF' /etc/passwd` tells you that there are seven fields used in the password file.)
- ☞ `NR` refers to which record (by default, line) you are currently at.



awk

Some examples of relational expressions:

```
$1 == 'lane' # does the first field equal the string 'lane'?  
$1 == $7    # are fields one and seven equal?  
NR > 1000   # have we processed more than 1000 records?  
NF > 10     # does this record have more than 10 fields?  
NF > 5 && $1 = 'me' # compound test  
/if/&&/up/   # does the record contain both strings if and up?
```



awk

You can also check a given field against a regular expression:

```
$1 ~ /D[Rr]\./ # does the first field contain a Dr. or DR.?  
$1 !~ /#/ # does the first field have a # in it?
```



awk

ACTIONS are specified with `{ }`. You can use semicolons to separate statements with the braces (although newlines work also). Popular statements are `print`, `if { } else { }`, and `system`.

`awk` is very powerful! Henry Spencer wrote an assembler in `awk`.



awk example scripts

```
{ print $1, $2 } # print the first two fields of each record

$0 !~ /^$/      # print all non-empty lines

$2 > 0 && $2 < 10 { print $2 } # print field 2 if it is 0 < $2 < 10

BEGIN {FS=','}
sum = 0} # sum field 3 and print the sum
{sum += $3}
END {print sum}
```



The `tr` utility

Allows you to delete, replace, or “squeeze” characters from standard input. The `-d` option deletes the characters specified in the first argument; `-s` squeeze removes all repetitions of characters in the first argument with a single instance of the character. The normal mode is to substitute characters from the first argument with characters from the second argument.



The tr utility

```
[langley@sophie 2006-Fall]$ cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1          localhost.localdomain localhost
128.186.120.8     sophie.cs.fsu.edu
127.0.0.1         a.as-us.falkag.net
127.0.0.1         clk.atdmt.com
[langley@sophie 2006-Fall]$ cat /etc/hosts | tr 'a-z' 'A-Z'
# DO NOT REMOVE THE FOLLOWING LINE, OR VARIOUS PROGRAMS
# THAT REQUIRE NETWORK FUNCTIONALITY WILL FAIL.
127.0.0.1          LOCALHOST.LOCALDOMAIN LOCALHOST
128.186.120.8     SOPHIE.CS.FSU.EDU
127.0.0.1         A.AS-US.FALKAG.NET
127.0.0.1         CLK.ATDMT.COM
```

