POWER TOOLS

# What's GNU, Part Three

By Jerry Peek

This month, let's dig into three related utilities — *mv*, *cp*, and *ln* — and some of the story behind their new features.

Back in *Unix Version 7*, *cp*, *mv*, and *ln* had no options: they simply copied, renamed, or linked one or more files to a destination. Now, *GNU cp* has more than twenty options, and *mv* and *ln* aren't far behind.

Let's look at some of the major new features of these fundamental utilities — and some minor and individual features, too — added by GNU programmers and others.

## Making Backups

Unix and Linux shells and utilities were written with the assumption that you know what you're doing. If you want to replace the kernel with your shopping list, *mv* won't ask, "Are you sure?" (although the superuser prompt # is an omnipresent reminder to be extra-careful). Many gurus prefer life this way.

Still, mistakes can happen. Many pre-GNU versions of *cp*, *mv*, and *ln* had a `-i` ("interactive") option that prompted for confirmation before it replaced existing destination files.

But, of course, that's only useful if you're running a utility interactively. From a script or a *cron* job, for instance, you might have prevented overwrites by using the almost-unknown utility *yes* to answer `n` to every prompt:

```
$ yes n | cp -i files* dest
```

(Though, if memory serves, at least one version of *cp* silently ignored `-i` if the standard input wasn't a terminal!)

Another way to avoid accidental overwrites was to step through source files one by one in a loop, making sure each destination file didn't exist before running *cp*. Multiple invocations of utilities like *cp* can be inefficient, though.

The GNU utilities *cp*, *mv*, and *ln* now have the option `--backup=`*`method`*, which makes a backup of each file that would otherwise be overwritten or removed. The backup can be made with one of four "version control" methods, listed in *Table One*. (Each method has two possible names.)

Another GNU option, `-b`, makes backups with the `existing` (or `nil`) method. (The `-V` and `--version-control` options — which are now deprecated for *ln*, at least — let you choose the method for `-b`.)

A `numbered` backup adds `.~`*n*`~` to the end of the original filename. If there are currently no backups, *n* is 1; otherwise, *n* is the lowest unused number. So, if you run `cp --backup=numbered` *`src dest`*, and *`dest`* already exists, then *cp* renames *dest* to *dest.~1~* before it copies *`src`* to *`dest`*. Also, for example, if *dest.~1~* and *dest.~23~* already exist, *cp* renames *dest* to *dest.~24~*.

A `simple` backup adds a tilde (`~`) to the end of the destination filename, then creates a copy at the destination name. Using this method, each file can have at most one backup. The option `-S` *SUFFIX* or `--suffix=`*`SUFFIX`* lets you choose a suffix other than a tilde. For instance, `mv -b --suffix=.bak foo bar` would make or replace a backup file named *bar.bak*. You can also store a global backup suffix in the environment variable `SIMPLE_BACKUP_SUFFIX`.

The sidebar "Back Me Up" shows a surprising way to backup a file and the sidebar "*ls* without backups" has a handy tip.

## Removing Backups

Let's look at a couple of ways to clean a directory full of backup files. A command like `rm* ~` or `rm* .~*~` removes all the simple or numbered backups at once. You could remove them all with an alias like this (in *csh* syntax):

```
alias deltemp 'rm *~'
```

For safety, you might add the *rm* `-i` option. You could also list all of the files and prompt, like this:

```
alias deltemp 'ls -l *~; \\
  echo -n "Remove all those files[ny]?";
\\
  if ($< =~ y*) rm *~'
```

Removing a range of numbered backup files — not all the files, just some — is probably easiest with the *Z Shell*'s numeric-range glob operator `<`*`m-n`*`>`.

### BACK ME UP

The *info* page for *GNU cp* shows a nice way to backup a file before editing (or for any other reason). Give the filename to *cp* as both the source and destination filename with the two options `--force` and `--backup` (or `-f` and `-b`), like this:

```
$ cp -fb foo foo
```

This creates a backup copy named, in this case, *foo~*. You can also set the backup method to make numbered backups.

**TABLE ONE:** Backup methods for *GNU cp, mv,* and *ln*

| METHOD | DESCRIPTION |
|---|---|
| `simple, never` | Always make simple backups. (Note that never is different than none.) |
| `existing, nil` | Make numbered backups of files that already have them and simple backups of others. |
| `numbered, t` | Always make numbered backups. |
| `none, off` | Never make backups. |

For instance, in *zsh*, the command `rm foo.~<1-99>~` removes any files in the range *foo.~1~*, *foo.~2~*, and so on, up to and including *foo.~99~*. If you aren't a *zsh* user, this *bash* function calls *zsh* to run just the one command:

```
deltemp()
{
  case $# in
  2) zsh -c "rm $1.~<$2>~" ;;
  *) echo "Usage: deltemp name suffix-range" ;;
  esac
}
```

Typing `deltemp foo 1-99` executes `zsh -c "rm foo.<1-99>"`. (The `zsh -c` option executes a single command taken from the next argument.)

Note that if you're making a lot of backup copies and you want to keep them, having a series of backup files in a directory can be less efficient than using a revision control system like *RCS* or *Subversion*.

For automated cleanup, nothing beats a *cron* job. The sidebar "Automated Cleanup" has some suggestions.

## Temporary Files

Automatic removal of backups gives you a new opportunity: delayed removal of *non*-backup files. That is, instead of removing some files with *rm*, you can rename them to look like backup files — and those files will be removed *en masse* sometime later by your backup-cleanup *cron* job. A setup like this can be the Linux shell's version of the *Macintosh* "Trash" or the *Windows* "Recycle Bin."

The *bash* script named *temp*, which you can get from http://www.linux-mag.com/downloads/2005-08/power/temp.txt, does this. It finds the highest-numbered existing backup file and renames a file with the next-higher suffix. It also uses *touch* to update the file's last-modification time so the file will stay around for the full "grace period" coded into the *cron* job.

## Copying Sparse Files

When *cp* reads a file, it makes crude checks for "sparse-ness." If it finds holes, it also makes holes in the destination file.

Some versions of *cp* haven't worked this way. You could copy a file that seemed to take few disk blocks and end up overflowing the destination filesystem. That's one reason this ability is useful. If you want to control it, though, use the *cp* option `--sparse=`*when*:

- If *when* is `auto` (the default), *cp* creates a sparse output file if the input file seems to be sparse. If the output file is non-regular — a tape drive, for instance — *cp* won't try to make it sparse. (`tar -S` is an efficient way to write sparse files to tape.)

How many ways can you CommuniGate?

With CommuniGate Pro Real-Time Communications Server, you can email, IM, video conference and make VoIP calls around the world and remain connected...always.

CommuniGate Pro

To download a free 30 day trial visit us at http://www.stalker.com

CommuniGate Pro Real-Time Communications Server Features

- Flexibility – Supports over 30 operating systems
- Security – Offers encryption, authentication & authorization
- Reliability – True 99.999% uptime

Visit us at Linux World, booth #916

STALKER SOFTWARE INC.

- Set *when* to `always` if the input file has long series of zero-bytes but is on a filesystem that doesn't support sparse files. If the output filesystem does support sparse files, *cp* makes a sparse file there. This is handy for restoring databases and other huge files from media where "sparse-ness" wasn't maintained.

- Setting *when* to `never` means that the output file is always non-sparse.

## Target Directories

When a command line has a series of filenames and a single directory name at the end, *cp*, *mv*, and *ln* copies (or moves or links) all of those files into that directory. That directory is called the *target directory*.

The GNU option `--target-directory=`*name* is another way to specify the target directory. You need `--target-directory` because of the way the *xargs* utility works: it appends a series of arguments to the end of a command-line. But that means that *cp*, *mv*, and *ln* can't have a target directory with *xargs* (unless you use the tricky *xargs* options `-i` or `--replace`).

### AUTOMATED CLEANUP

Backup files can eventually swamp your directories. A nightly or weekly *cron* job running *find* can be the perfect way to clean them up. (For more on *find*, see the September 2002 column "A Very Valuable Find," online at http://www.linux-mag.com/2002-09/power_01.html.)

Your job could remove only those backup files that belong to you, or it could skip backup files with recent timestamps, since those files have probably just become backup files. (When *cp* makes a backup, it updates the last-modified timestamp.)

Here's a *find* command you can run from a *cron* job. A personal *cron* job is run from your home directory, so `.` in the first argument starts searching from your home directory:

```
find . \( -regex '^.*\.~[0-9]+~' \
  -o -name '*~' -o -name '.?*~' \)
  -type f -mtime +7 -exec rm -f {} \;
```

The `-regex` test uses an extended regular expression to find numbered backup files, including "hidden" filenames that start with a dot (as in *.cshrc*). `-name` tests find both regular and hidden filenames ending with a tilde; these use a shell wildcard pattern instead of a regular expression.

The `-exec rm -f {} \;` removes the files one-by-one, which can be inefficient if there are lots of files to remove. You might want to replace it with `-print0` and pipe the output of *find* to `xargs -r0 rm -f`. The commands `find -print0` and `xargs -0` are explained in the May 2005 column "Filename Trouble," available online at http://www.linux-mag.com/2005-05/power_01.html. The *xargs* option `-r` keeps it from running *rm* if there are no files to remove.

### *ls* without backups

The GNU *ls* option `-B` lists a directory, omitting the simple and numbered backup files. This is a great help in directories with lots of backup files.

Here's an example. Your file *edited* lists all files that you edited yesterday. Before you start work today, you want to copy all of those files into the directory */backups/2005-08-05*. Here's how:

```
$ xargs cp –target-directory=/back-
ups/2005-08-05 < edited
```

(You could also do that with command substitution — the shells' backquote or `$()` operators — but those can fail if the argument list is too long or if filenames have special characters. Using `xargs -0` can be even more reliable.)

## Stripping Trailing Slashes

One little-known part of the *POSIX* standard can cause trouble with symbolic links. If you use filename completion on a symbolic link and that link is a source argument to *mv*, look what happens:

```
$ ls -ld a b c
ls: c: No such file or directory
drwxr-xr-x  2 jpeek ... a
lrwxrwxrwx  1 jpeek ... b -> a
$ mv b/ c
mv: cannot move `b/' to `c': Not a direc-
tory
```

In this case, you can either remove the trailing slash by backspacing over it on the command line or you can add the `--strip-trailing-slashes` option to make *mv* ignore the slash. There's more information in the *mv info* page.

## There's More...

Read the *info* page topics about handling links, device files, and more — especially during recursive copies. These options are self-explanatory, but worth knowing about.

Next month's column changes topics, but there will be more about what's GNU in modern versions of perennial utilities in the first part of next year.

*Jerry Peek is a freelance writer and instructor who has used Unix and Linux for over 20 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.*