

## A Batch Job to Add New User IDs

Contributed by duke

A common system administration task is to add new users. In large installations, such as central computing servers at universities or in large companies, adding users is often best performed as a "batch" job, one that is automated with scripts. Consider, for example, the start of a new semester at a large public university, where there are hundreds, if not thousands, of new students. Creating accounts by hand would be impossible, so we need to automate the task. User accounts on Unix systems must have a unique User ID number (UID). Unix distinguishes users with the UID, and it uses it to enforce its file and process permissions. When adding new users, therefore, we have to assign each new user their own unique, previously unused, UID number.

When assigning new UIDs, one approach might be to just take the highest existing UID number and increment from there. But UID numbers, while having a fairly large maximum value, are not infinite. In particular, some systems limit the maximum value to that of a 16-bit unsigned integer, 65535. One can easily imagine that, in a large-scale environment, the "take the highest value and increment" approach would run out of numbers after a few years.

One must also consider that over the course of time user accounts will be deleted. For example, as students graduate and enter the work force (or go on to grad school somewhere else, as the case may be), their accounts need to be deleted. This entails, at least, removing all their files and removing their entries from the password file for the system.

Given these considerations, a smarter approach is to note that there will be gaps within the range of allocated UID numbers. When new accounts are created, they should be given unused UID numbers allocated from an existing gap in the UID space. The following script implements this idea; it could be used as one component of a larger suite of scripts for adding users. The script's usage is:

```
newuids [-c N]
```

The program's default action is to print the first unused UID from those available in the password file, `/etc/passwd`. If given the option `-c` with a count `N`, it prints `N` unused UID numbers.

The program begins with the `#!/` line to invoke `ksh`, and an initial comment documenting the basic usage:

```
#!/bin/ksh

# newuids --- print one or more unused uids
#
# usage:
#   newuids [-c N]
#   -c N      print N unused uids
```

Next comes some variable assignments, and a trap command to clean up the temporary file when the program exits:

```
PASSWD=${PASSWD:-/etc/passwd}
TMPFILE=/tmp/uidlist$$
```

```
trap 'rm -f $TMPFILE' EXIT HUP TERM # clean up on exit or signal
```

Then comes the argument parsing. We use the built-in `getopts` command to process the one possible option. Doing so is a bit of overkill for just one option, but it allows for future growth as more options may be added over time. Planning ahead is always a good idea.

```
count=1 # how many uids to print
```

```
# parse arguments, let ksh issue diagnostics
# and exit if need be
while getopts "c#" opt
do
    case $opt in
    c) count=$OPTARG ;;
    esac
done
```

The Unix `/etc/passwd` file describes the system's users, one user per line. Each line consists of seven fields: the username, the encrypted password field, the UID, the Group ID (GID), the user's full name, the home directory (the initial value of `$HOME` for that user), and finally, the login shell. The fields are separated by colons. For example:

```
arnold:xyzy42:2076:10:Arnold D. Robbins:/home/arnold:/bin/ksh
```

In order to find unused UID numbers, we have to extract a sorted list of existing UID numbers from the password file. Extracting the UID numbers is done with `awk`, and sorting the list is done with `sort`. The sorted list is saved in the file named by `$TMPFILE`:

```
awk -F: '{ print $3 }' $PASSWD | # generate list of uids
sort -n -u > $TMPFILE          # sort numerically, remove duplicates
```

We then read the sorted list into the indexed array `uidlist`, and save the total number of elements in the variable `totalids`:

```
set -A uidlist $(< $TMPFILE)    # save in indexed array

totalids=${#uidlist[*]}        # total number of uids
```

At this point, all the setup is complete. All that remains is to loop over the array, finding a pair of entries that are noncontiguous. When one is found, the script prints all the values between the first entry in the pair and the second value. The variable `count` is decremented each time, to ensure that the program won't print more than the requested number of unused UIDs:

```
# loop over ids, finding non-contiguous ones
for ((i = 2; i <= totalids; i++))
do
    if (( uidlist[i-1] + 1 != uidlist[i] ))
    then
        for ((j = uidlist[i-1] + 1; j < uidlist[i]; j++))
        do
            print $j
            if (( --count == 0 ))
            then
                break 2
            fi
        done
    fi
done
```

The first thing to notice about this loop is the heavy use it makes of the Korn shell's `((...))` construct. In particular, variables inside `((...))` don't need dollar signs to get their values, and it is possible to use numeric expressions for subscripts. (The latter is always true; it's not limited to just being inside `((...))`.) The expression `--count` decrements `count` before testing its value. If `count` starts out at 1, it becomes 0. With higher initial values, it counts down until it becomes 0. The next thing to notice is the use of `break 2`, which breaks out of the outermost enclosing for-loop. This is a convenient feature not available in languages like C, C++, or `awk`. It nicely avoids use of the dreaded `goto` (which doesn't exist in Bourne-derived shells).

That's it! Here is the script again, in its entirety, should you wish to cut and paste it from your browser:

```
#!/bin/ksh

# newuids --- print one or more unused uids
#
# usage:
#   newuids [-c N]
#   -c N      print N unused uids

PASSWD=${PASSWD:-/etc/passwd}
TMPFILE=/tmp/uidlist$$

trap 'rm -f $TMPFILE' EXIT HUP TERM # clean up on exit or signal

count=1 # how many uids to print

# parse arguments, let ksh issue diagnostics
# and exit if need be
```

```

while getopts "c#" opt
do
    case $opt in
    c) count=$OPTARG ;;
    esac
done

awk -F: '{ print $3 }' $PASSWD | # generate list of uids
sort -n -u > $TMPFILE          # sort numerically, remove duplicates

set -A uidlist $(< $TMPFILE)   # save in indexed array

totalids=${#uidlist[*]}       # total number of uids

# loop over ids, finding non-contiguous ones
for ((i = 2; i <= totalids; i++))
do
    if (( uidlist[i-1] + 1 != uidlist[i] ))
    then
        for ((j = uidlist[i-1] + 1; j < uidlist[i]; j++))
        do
            print $j
            if (( --count == 0 ))
            then
                break 2
            fi
        done
    fi
done

```

It is possible to enhance this script. For example, consider adding an option specifying the smallest allowed UID number. This might be useful for centralized servers where the policy is that ranges of UID numbers are allocated to different departments or schools.

Another enhancement might be to use the enhanced capabilities of the Korn shell's getopts command to allow long options and/or produce nice output from newuids --man.

Arnold Robbins is a professional programmer and technical author who has worked with Unix systems since 1980.

[Return to the Linux DevCenter.](#)

[Search Linux](#)  
[Tagged Articles](#)

Be the first to post this article to [del.icio.us](http://del.icio.us)  
[Sponsored Resources](#)

- \* [Inside Port 25](#)
- \* [Inside Lightroom](#)

[Related to this Article](#)

[The FreeBSD 6.2 Crash Course](#) [The FreeBSD 6.2 Crash Course](#)  
 by Jem Matzan  
 May 2007  
 \$9.99 USD

[Managing RPM-Based Systems with Kickstart and Yum](#) [Managing RPM-Based Systems with Kickstart and Yum](#)  
 by Ethan McCallum  
 March 2007  
 \$9.99 USD  
 Advertisement

[Contact Us](#) | [Advertise with Us](#) | [Privacy Policy](#) | [Press Center](#) | [Jobs](#)

Copyright © 2000-2006 O'Reilly Media, Inc. All Rights Reserved.

All trademarks and registered trademarks appearing on the O'Reilly Network are the property of their respective owners.

For problems or assistance with this site, email `'`); if (display) { document.write(display); } else { document.write(name + '@' + domain); } document.write(" + ending); // -->