Bash (/tag/bash) HOW-TOs (/tag/how-tos) Shell Scripting (/tag/shell-scripting)

Bash Shell Games: Let's Play Go Fish!

by Dave Taylor (/users/dave-taylor) on July 30, 2019

How to begin developing a computer version of the popular card game.

Between the previous 163 columns I've written here in *Linux Journal* and the dozens of games I programmed and explored during the creation of my *Wicked Cool Shell Scripts* book, I've written a lot of Bash shell games. The challenge is to find one that's simple enough where a shell script will work, but isn't so simple that it ends up being only a half-dozen lines.

Magic 8-Ball is a perfect example. It turns out that the entire "predict the future" gizmo was really just a 20-sided die floating in dark purple fluid. So an array of 20 possible values and a random number selector and boom—you've got a magic 8-ball script:

Recent Articles

(/content/open-source-good-how-can-it-do-good)
Open Source Is Good, but How Can It Do Good? (/content/open-source-good-how-can-it-do-good)
Glyn Moody (/users/glyn-moody)

(/content/episode-24-chat-about-redis-labs-podcast-transcript)
Reality 2.0 Episode 24: A Chat About Redis Labs (Podcast Transcript) (/content/episode-24-chat-

```
#!/bin/sh

# magic 8 ball. Yup. Pick a random number, output message

# messages harvested from the Wikipedia entry

answers=("It is certain." "It is decidedly so."
  "Without a doubt." "Yes – definitely."
  "You may rely on it." "As I see it, yes." "Most likely."
  "Outlook good." "Yes." "Signs point to yes."
  "Reply hazy, try again." "Ask again later."
  "Better not tell you now." "Cannot predict now."
  "Concentrate and ask again." "Don't count on it."
  "My reply is no." "My sources say no."
  "Outlook not so good." "Very doubtful.")

echo "Oh! Magic 8 Ball, Please Tell Me True..." ; echo ""
/bin/echo -n "What is your question? "
read question

answer=$(( $RANDOM % 20 ))

echo ""
echo "I have looked into the future and I say: "
echo "     ${answers[$answer]}" ; echo ""

exit 0
```

Let's do a quick run to see if I'm the most popular *LJ* writer:

```
$ sh magic8.sh
Oh! Magic 8 Ball, Please Tell Me True...

What is your question? Am I the most popular LJ writer?

I have looked into the future and I say:
     My reply is no.
```

Ouch, that's harsh. I write the darn divination program, and it just drops a brick on my foot. Yeesh.

More seriously, Magic 8 Ball is too simple to make an interesting shell script. By contrast, *Call of Duty* is way too complex, even if I did a version with text output instead of gorgeously rendered 3D graphics.

Card Game Function Library

That's why card games prove to be good as programming challenges or exercises: the core mechanism of a 52-card random deck is pretty straightforward, so it's all about the actual cardplay.

Not only that, but as I've written before about card games as shell scripts, I already have a handy set of functions to create, shuffle and display cards out of a deck. If you want to rummage in the archives, I've tackled *Acey-Deucey*, *Baccarat* and some bits and pieces of *Cribbage*.

about-redis-labs-podcast-transcript)

Katherine Druckman (/users/katherine-druckman)

(/content/where-internet-gets-real)

Where the Internet Gets Real (/content/where-internet-gets-real)

Doc Searls (/users/doc-searls)

The DevOps Issue

(/content/devops-issue)

(/content/devops-issue)

Bryan Lunduke (/users/bryan-lunduke)

(/content/continuous-integrationcontinuous-development-foss-tools)

Continuous Integration/Continuous Development with FOSS Tools (/content/continuous-integrationcontinuous-development-foss-tools)

Quentin Hartman (/users/quentin-hartman)

(/content/bash-shell-games-lets-play-go-fish)

Bash Shell Games: Let's Play Go Fish! (/content/bash-shell-games-lets-play-go-fish)

Dave Taylor (/users/dave-taylor)

Related Articles

In order to jump right into the new game that I'm going to describe how to build, *Go Fish!*, let's steal the following functions from my earlier scripts:

- `initializeDeck`
- `shuffleDeck`
- `pickCard`
- `showCard`

I've uploaded the script library to my AskDaveTaylor site, so you can grab it here (https://www.askdavetaylor.com/playing-card-library/) if you like.

With those functions all available, let's start writing a *Go Fish!* game.

Heading to the Fishing Hole

If you don't have kids, it might have been a while since you've played the simple game of *Go Fish!*. I would categorize it as a memorization game, because if you can remember what the other player has asked, you'll generally win the game.

Go Fish! is pretty simple. Each player starts with seven cards, and they take turns asking the other player if they have one or more of a particular card rank. If they do, they hand over all of their cards of that rank. If they do not, then you "go fish" and take the top card from the deck.

Once you have four of a kind, you immediately place those in front of you. If you ever have no cards in your hand, you immediately pick two from the deck and play proceeds. The game ends when there are no cards left in either person's hand and none in the deck. The winner is the one with more sets at the end of the game. It's pretty simple.

Note: the game becomes a lot more interesting with three or four players. Try it!

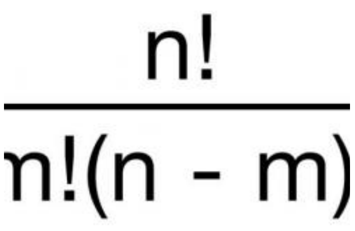
To get started, here's an easy way to include a set of functions from another file:

```
cardlib="playing-card-library.sh"

if [ ! -f $cardlib ] ; then
  echo "Can't find the playing card library $cardlib"
  exit 1
fi

. $cardlib
```

The conditional test before the "." source statement might be redundant, but why not have a nice error message in the case that the cardlib file isn't found?

The source statement (with slightly different syntax in different shells) is interesting. It reads the specified file as if its contents were part of the current file. This means any functions, any variables, anything that's in the file is incorporated into the current shell, not a subshell (as would happen if you used `sh $cardlib`, for example).

Now that the functions are defined, it's time to deal seven cards to both the computer player and the user. To track all this, I'm going to use two arrays: `myhand` and `yourhand`. The shell dynamically resizes arrays as needed, which is great for *Go Fish!*, because you could end up having a lot more than seven cards during the game:



(/content/cribbage-calculating-hand-value)

Cribbage: Calculating Hand Value (/content/cribbage-calculating-hand-value)

by Dave Taylor

(/content/cribbage-sorting-your-hand)

Cribbage: Sorting Your Hand (/content/cribbage-sorting-your-hand)

by Dave Taylor

(/content/counting-cards-cribbage)

Counting Cards: Cribbage (/content/counting-cards-cribbage)

by Dave Taylor

Corporate Patron

```
i=1

while [ $i -lt 8 ] ; do
  myhand[$i]=${newdeck[$i]}
  yourhand[$i]=${newdeck[$(( $i + 7 ))]}
  i=$(( $i + 1 ))
done
```

There are a lot of punctuation symbols in this line:

```
yourhand[$i]=${newdeck[$(( $i + 7 ))]}
```

Unwrap it step by step, and you'll remember that any array reference must be `${name[x]}`, which is half the complexity. The other half is the `$(( equation ))` notation to do some basic math without a subshell. Basically, this is like the worst dealer in the world, dealing cards 1–7 to the computer and 8–14 to the player. But if it's shuffled...

The `showCard` function (part of `cardlib`) is a bit clunky, because it doesn't actually output the card name, it just preloads global variable `cardname` with the correct value. So here's how to show a hand:

```
echo computer hand:
showCard ${myhand[1]} ; echo "  $cardname"
showCard ${myhand[2]} ; echo "  $cardname"
showCard ${myhand[3]} ; echo "  $cardname"
showCard ${myhand[4]} ; echo "  $cardname"
showCard ${myhand[5]} ; echo "  $cardname"
showCard ${myhand[6]} ; echo "  $cardname"
showCard ${myhand[7]} ; echo "  $cardname"
```

Running this, it's clear that I might need to sort the cards by rank to make the game easier to play:

```
$ sh gofish.sh
computer hand:
  5 of Diamonds
  9 of Hearts
  4 of Clubs
  7 of Diamonds
  8 of Hearts
  K of Hearts
  K of Clubs
your hand:
  5 of Hearts
  9 of Diamonds
  Q of Diamonds
  2 of Spades
  6 of Clubs
  2 of Diamonds
  Q of Spades
```

Since the player won't ever have to specify an individual card, I think I can get away without sorting the hand, so I'll leave that as a task for you, dear reader, when you start fiddling with the code.

Won't need to specify a card? Right. You'll ask the computer if it has any, say, "twos" by entering a "2" at the prompt. In fact, to make this interesting, let's let the player potentially cheat by having the computer ask if they have a specific card, rather than automatically just taking it out of the hand.

I'm imagining a play sequence like this:

```
Your hand:
  5 of Hearts
  9 of Diamonds
  Q of Diamonds
  2 of Spades
  6 of Clubs
  2 of Diamonds
  Q of Spades

You go first. You ask me if I have: 3
** You don't have any 3s so you can't ask for that!
You ask me if I have: 2
** I do not. Go fish.
(( you pick up the 9 of Hearts ))

My turn. Do you have any 7s? (yes/no):
```

With that in mind, let's stop here and pick up this coding project next time. See you then, and in the meantime, find a youngling and play a few games of *Go Fish!* to see the (admittedly fairly minimal) nuances of the game.

Dave Taylor has been hacking shell scripts on UNIX and Linux systems for a really long time. He's the author of *Learning* *Unix for Mac OS X* and *Wicked Cool Shell Scripts*. You can find him on Twitter as @DaveTaylor, and you can reach him through his tech Q&A site: Ask Dave Taylor (https://www.askdavetaylor.com).

No comments yet. Be the first! (https://www.linuxjournal.com/content/bash-shell-games-lets-play-go-fish#disqus_thread)