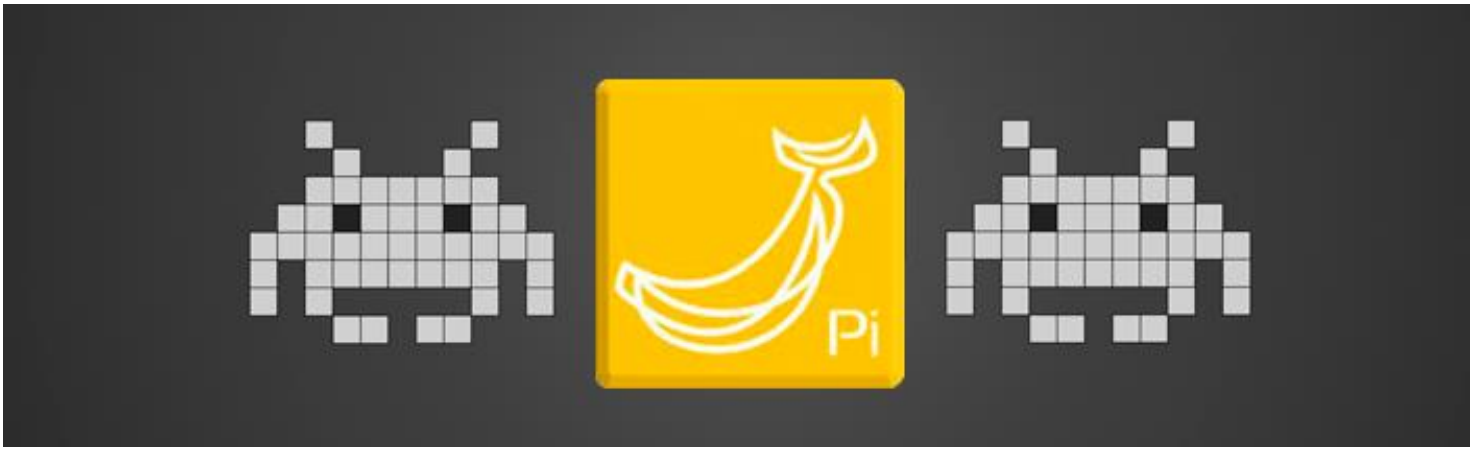
Imagine você jogando os games mais conhecidos com seu design baseado em linhas de comando Linux. Vamos testar?

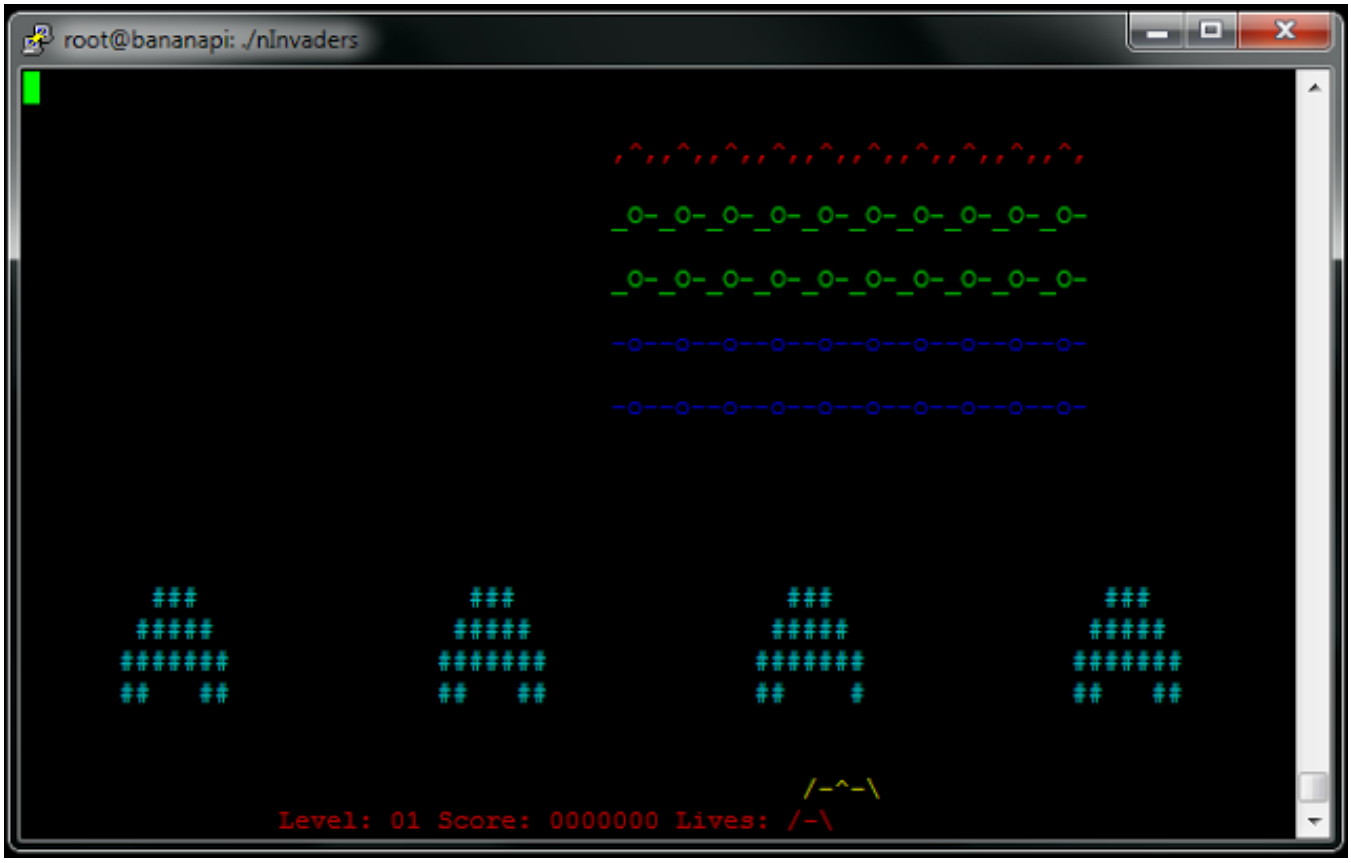


Jogos de Linha de Comando

Jogos de linha de comando parecem ser ainda mais antigos do que jogos retro de 8 bits que você já achava que eram muito antigos.

Jogos de linha de comando são executados em uma linha de comando e os gráficos são feitos de caracteres de entrada de texto que podem ser digitados. Mas, por que jogar este tipo de jogo? Bem, eles são interessantes e alguns jogos podem ser realmente divertidos.

nInvaders - Space Invaders



Instalação

```
apt-get install ninvaders
```

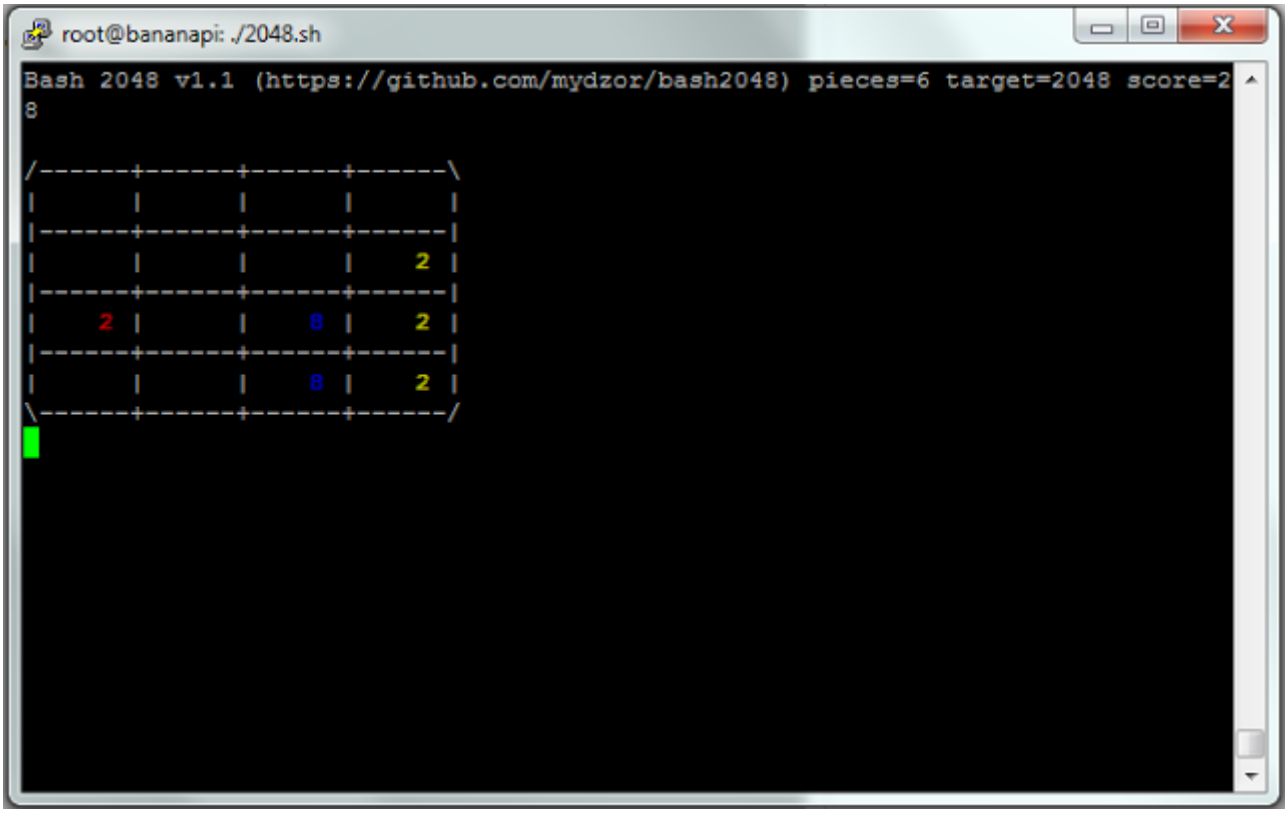
Execute

```
cd/usr/games
./nInvaders
```

Sair

```
control + c
```

Bash 2048 - 2048 no .sh



Altere para o diretório de instalação:

```
cd /usr/games
```

Crie um novo arquivo 2048.sh

```
nano 2048.sh
```

Nesse arquivo que você acabou de criar, cole o conteúdo abaixo:

```
#!/usr/bin/env bash

#important variables
declare -ia board # array that keeps track of game status
declare -i pieces # number of pieces present on board
declare -i score=0 # score variable
declare -i flag_skip # flag that prevents doing more than one operation on
# single field in one step
declare -i moves # stores number of possible moves to determine if player lost
# the game
declare ESC=$'\e' # escape byte
declare header="Bash 2048 v1.1 (https://github.com/mydzor/bash2048)"

declare -i start_time=$(date +%s)

#default config
declare -i board_size=4
declare -i target=2048
declare -i reload_flag=0
declare config_dir="$HOME/.bash2048"

#for colorizing numbers
declare -a colors
colors[2]=33 # yellow text
colors[4]=32 # green text
colors[8]=34 # blue text
colors[16]=36 # cyan text
colors[32]=35 # purple text
colors[64]="33m\033[7" # yellow background
colors[128]="32m\033[7" # green background
colors[256]="34m\033[7" # blue background
colors[512]="36m\033[7" # cyan background
colors[1024]="35m\033[7" # purple background
colors[2048]="31m\033[7" # red background (won with default target)

exec 3>/dev/null # no logging by default

trap "end_game 0 1" INT #handle INT signal

#simplified replacement of seq command
function _seq {
local cur=1
local max
local inc=1
case $# in
1) let max=$1;;
2) let cur=$1
let max=$2;;
3) let cur=$1
let inc=$2
let max=$3;;
esac
while test $max -ge $cur; do
printf "$cur "
let cur+=inc
done
}

# print currect status of the game, last added pieces are marked red
function print_board {
clear
printf "$header pieces=$pieces target=$target score=$score\n"
printf "Board status:\n" >&3
```

```
printf "\n"
printf '/------'
for l in $(_seq 1 $index_max); do
printf '+------'
done
printf '\\\n'
for l in $(_seq 0 $index_max); do
printf '|'
for m in $(_seq 0 $index_max); do
if let ${board[l*$board_size+m]}; then
if let '(last_added==(l*board_size+m))|(first_round==(l*board_size+m))'; then
printf '\033[1m\033[31m %4d \033[0m|' ${board[l*$board_size+m]}
else
printf "\033[1m\033[${colors[${board[l*$board_size+m]}]}m %4d\033[0m |" ${board[l*$board_size+m]}
fi
printf " %4d |" ${board[l*$board_size+m]} >&3
else
printf ' |'
printf ' |' >&3
fi
done
let l==$index_max || {
printf '\n|------'
for l in $(_seq 1 $index_max); do
printf '+------'
done
printf '|\n'
printf '\n' >&3
}
done
printf '\n\\------'
for l in $(_seq 1 $index_max); do
printf '+------'
done
printf '/\n'
}

# Generate new piece on the board
# inputs:
# $board - original state of the game board
# $pieces - original number of pieces
# outputs:
# $board - new state of the game board
# $pieces - new number of pieces
function generate_piece {
while true; do
let pos=RANDOM%fields_total
let board[$pos] || {
let value=RANDOM%10?2:4
board[$pos]=$value
last_added=$pos
printf "Generated new piece with value $value at position [$pos]\n" >&3
break;
}
done
let pieces++
}

# perform push operation between two pieces
# inputs:
# $1 - push position, for horizontal push this is row, for vertical column
# $2 - recipient piece, this will hold result if moving or joining
# $3 - originator piece, after moving or joining this will be left empty
# $4 - direction of push, can be either "up", "down", "left" or "right"
```

```
# $5 - if anything is passed, do not perform the push, only update number
# of valid moves
# $board - original state of the game board
# outputs:
# $change - indicates if the board was changed this round
# $flag_skip - indicates that recipient piece cannot be modified further
# $board - new state of the game board
function push_pieces {
case $4 in
"up")
let "first=$2*$board_size+$1"
let "second=($2+$3)*$board_size+$1"
;;
"down")
let "first=(index_max-$2)*$board_size+$1"
let "second=(index_max-$2-$3)*$board_size+$1"
;;
"left")
let "first=$1*$board_size+$2"
let "second=$1*$board_size+($2+$3)"
;;
"right")
let "first=$1*$board_size+(index_max-$2)"
let "second=$1*$board_size+(index_max-$2-$3)"
;;
esac
let ${board[$first]} || {
let ${board[$second]} && {
if test -z $5; then
board[$first]=${board[$second]}
let board[$second]=0
let change=1
printf "move piece with value ${board[$first]} from [$second] to [$first]\n" >&3
else
let moves++
fi
return
}
return
}
let ${board[$second]} && let flag_skip=1
let "${board[$first]}==${board[second]}" && {
if test -z $5; then
let board[$first]*=2
let "board[$first]==$target" && end_game 1
let board[$second]=0
let pieces-=1
let change=1
let score+=${board[$first]}
printf "joined piece from [$second] with [$first], new value=${board[$first]}\n" >&3
else
let moves++
fi
}
}

function apply_push {
printf "\n\ninput: $1 key\n" >&3
for i in $(_seq 0 $index_max); do
for j in $(_seq 0 $index_max); do
flag_skip=0
let increment_max=index_max-j
for k in $(_seq 1 $increment_max); do
let flag_skip && break
```

```
push_pieces $i $j $k $1 $2
done
done
done
}

function check_moves {
let moves=0
apply_push up fake
apply_push down fake
apply_push left fake
apply_push right fake
}

function key_react {
let change=0
read -d '' -sn 1
test "$REPLY" = "$ESC" && {
read -d '' -sn 1 -t1
test "$REPLY" = "[" && {
read -d '' -sn 1 -t1
case $REPLY in
A) apply_push up;;
B) apply_push down;;
C) apply_push right;;
D) apply_push left;;
esac
}
} || {
case $REPLY in
k) apply_push up;;
j) apply_push down;;
l) apply_push right;;
h) apply_push left;;
esac
}
}

function save_game {
rm -rf "$config_dir"
mkdir "$config_dir"
echo "${board[@]}" > "$config_dir/board"
echo "$board_size" > "$config_dir/board_size"
echo "$pieces" > "$config_dir/pieces"
echo "$target" > "$config_dir/target"
# echo "$log_file" > "$config_dir/log_file"
echo "$score" > "$config_dir/score"
echo "$first_round" > "$config_dir/first_round"
}

function reload_game {
printf "Loading saved game...\n" >&3

if test ! -d "$config_dir"; then
return
fi
board
```

Defina as permissões:

```
chmod 777 2048.sh
```

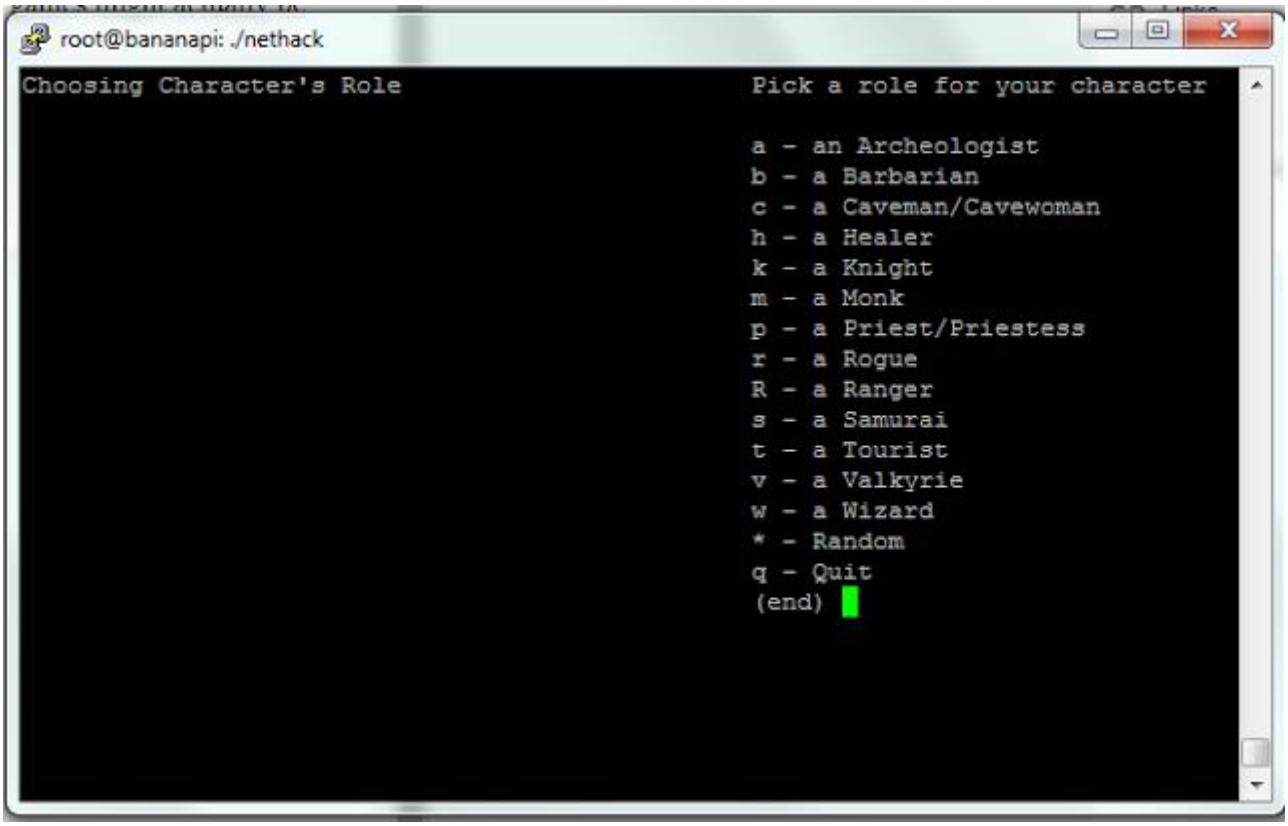
Execute o 2048.sh

```
./2048.sh
```

Saia

```
control + c
```

NetHack - Dungeon Explorer

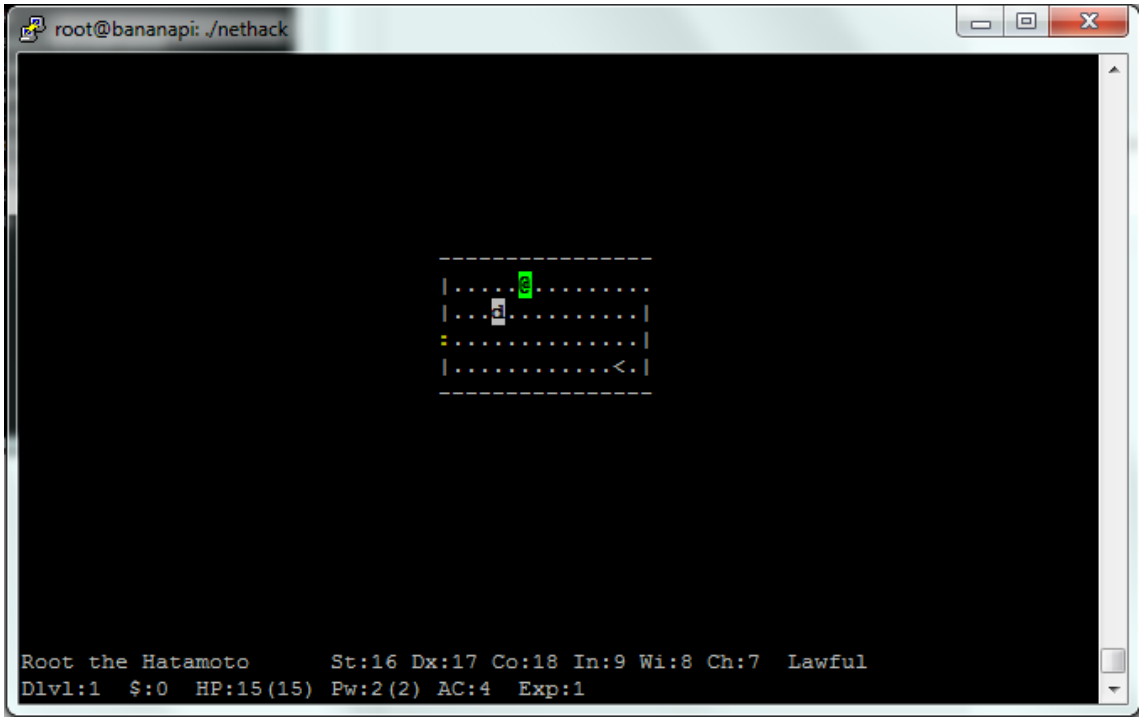


Instale

```
apt-get install nethack-console
```

Localize e execute:

```
cd /usr/games
./nethack
```



Mais?

É claro que há muito mais jogos divertidos que podem ser executados no terminal Linux. Se você quiser pode encontrar outros jogos no google e reproduzi-los!

Fonte: http://projectbananapi.blogspot.com.br/2015/07/play-ninvaders-2048-nethack-and-other.html