Ivan Constantin, https://www.flickr.com/photos/ivan70s/

Unleash your inner console cowboy

Kenneth Geisshirt
kg@realm.io

Realm Inc.
@realm
http://github.com/Realm/
http://realm.io/

Today's goal

- Present bash as a productivity tool
    - stop using the mouse ~~🐁~~
- Write scripts to automate your work
- Begin to use advanced tools in your daily work

Become a console cowboy

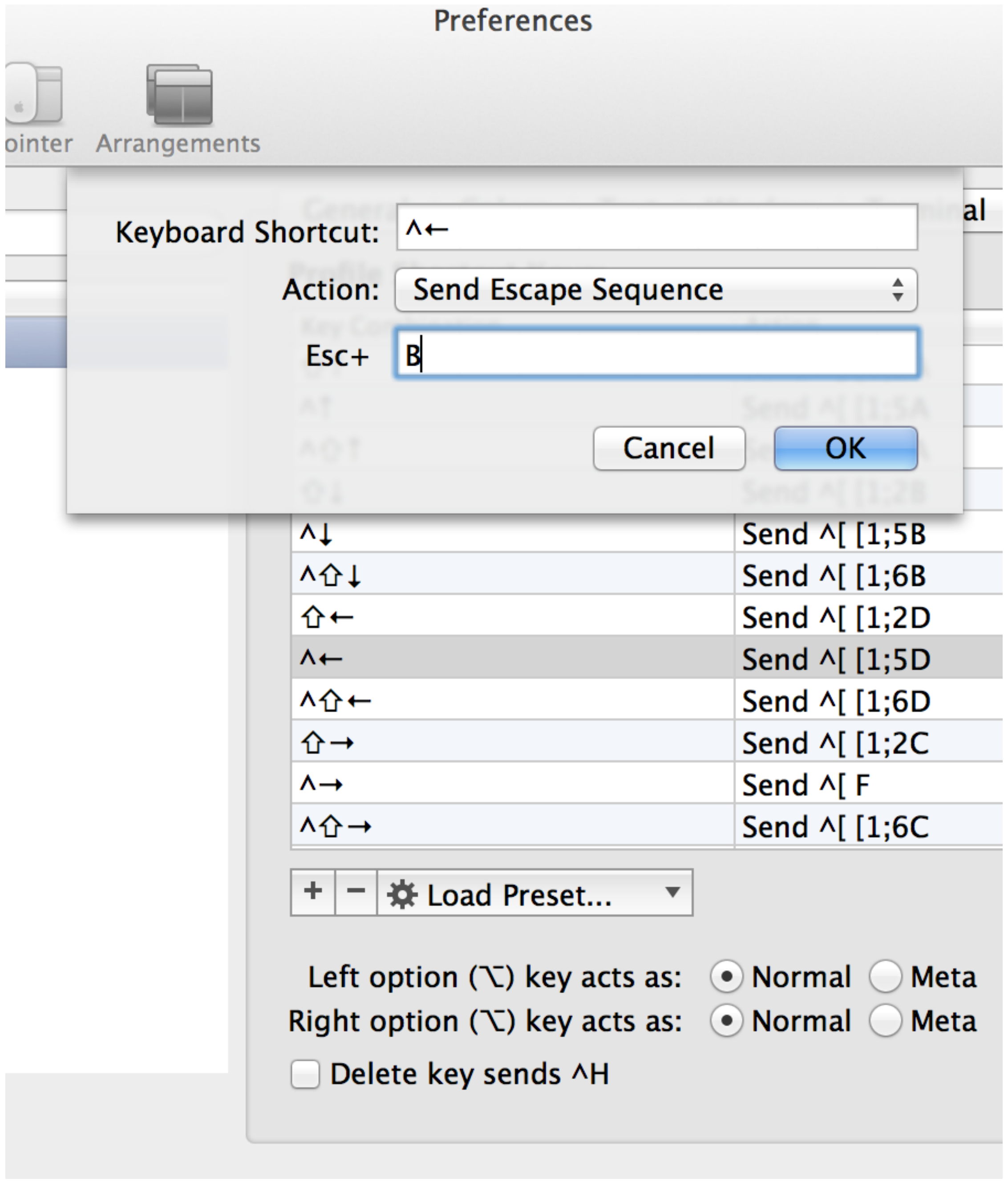
Agenda

- The terminal and the shell
- Basic usage of bash
- Living on the command-line
- Useful utilities
- Scripting
- Home brew
- Tools for developers
- git
- Xcode

The shell

Which terminal?

- iTerm2 is much better
  - Easier to change tab (⌘ left + right, CTRL+TAB)
  - Change Desktop navigation to ⌥ left + right
  - Add CTRL left + right to iTerm2 preferences
  - Keeps SSH connection alive
- http://iterm2.com/



Keyboard Shortcut: ^←
Action: Send Escape Sequence
Esc+ B

^↓	Send ^[[1;5B
^⇧↓	Send ^[[1;6B
⇧←	Send ^[[1;2D
^←	Send ^[[1;5D
^⇧←	Send ^[[1;6D
⇧→	Send ^[[1;2C
^→	Send ^[F
^⇧→	Send ^[[1;6C

Left option (⌥) key acts as: [✓ Normal] Meta
Right option (⌥) key acts as: [✓ Normal] Meta
☐ Delete key sends ^H

Which shell?

Stephen R. Bourne (Bell lab) introduced the shell to UNIX in 1977

OS X comes with many shells

- bash, csh, ksh, sh, tcsh, and zsh

Parée, https://www.flickr.com/photos/pareeerica/

Since 10.3, bash has been the default shell

OS X 10.11.1 carries bash 3.2.57 (2014-11-07)

Home brew has many great bash related packages

Redirection

UNIX idioms

- a tool should do one thing but do it well
- text is the universal data format

} Output of one utility is input for the next

Bash implements redirection:

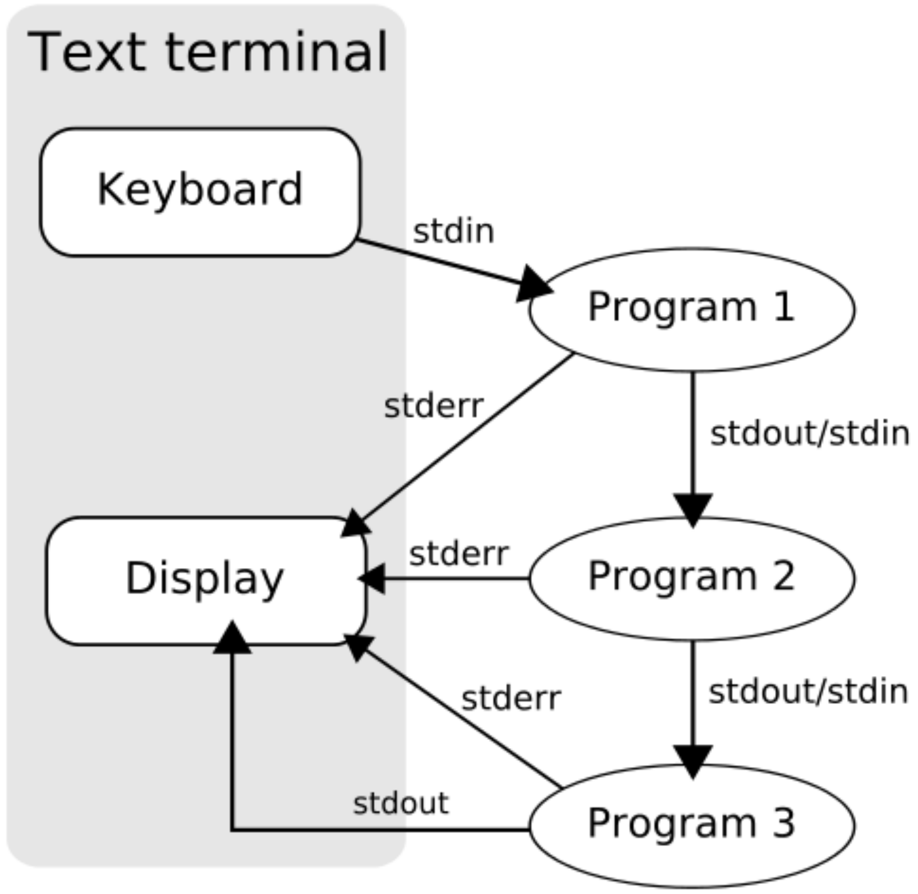
- stdout to file: >
- stdin from file <
- append stdout to file: >>
- stderr to stdout: `2>&1`

Examples:

```
echo "Hello" > hello
cat < hello
echo "World" >> hello
clang notfound.m > error 2>&1
```

Pipes

- Introduced to UNIX by Douglas McIlroy in 1973
- Pipes are the glue in UNIX component based programming (aka shell scripting)
- Powerful idiom for stream processing
- The character `|` is used by all known shells



"Pipeline" by TylzaeL - Licensed under Public domain via Wikimedia Commons
http://commons.wikimedia.org/wiki/File:Pipeline.svg#mediaviewer/File:Pipeline.svg

Examples

```
lsof | grep ^Keynote | wc -l
ifconfig | grep -e ^[a-z] | cut -f1 -d:
```

Configuration

- `$HOME/.bash_profile` and `$HOME/.bashrc` are your personal configuration
- `alias` - useful for often used options
- Setting prompt (`PS1`) and search path (`PATH`)
- Reload configuration: `source ~/.bash_profile`

.bash_profile

```
# Ignore a few commands in history
export HISTIGNORE="pwd:ls:ls -l:cd"
# don't put duplicate lines in the history. See bash(1) for more options
# don't overwrite GNU Midnight Commander's setting of `ignorespace'.
HISTCONTROL=$HISTCONTROL${HISTCONTROL+:}ignoredups

# Bash completion
if [ -f $(brew --prefix)/etc/bash_completion ]; then
    . $(brew --prefix)/etc/bash_completion
fi

# Prompt - including git
PS1='\u@\h:\w$(__git_ps1 " (%s)") \$ `

# Color ls etc.
alias ls="ls -G"
alias ll="ls -l"

# https://xkcd.com/149/
alias fuck='sudo $(history -p \!\!)' # rerun as root
```

Keyboard short-cuts

- Bash uses Emacs bindings by default 😄
  (`help bind` for details)

Movement

- CTRL-a beginning of line
- CTRL-e end of line
- CTRL-← One word left
- CTRL-→ One word right

Remap CTRL-←/→

Cut-n-paste

- CTRL-space mark
- ESC ⬅ BACK Delete word
- CTRL-d Delete character
- CTRL-_ undo
- CTRL-k delete until end of line
- CTRL-y yank from kill-ring

History

- Bash stores your command history
  - `history` - show latest commands
  - `!!` - run last command
  - `!`*`number`* - run command *`number`* again
  - CTRL-r - search in history
- Exclude commands from history:
  - `export HISTIGNORE=“pwd:ls:ls -l:cd”`
- Exclude duplicates from history:
  - `export HISTCONTROL:ignoredups`

Completion

- Use `TAB` to let Bash complete as much as possible
- Use `TAB+TAB` to show possible completions
- Bash has programmable completion → you can specify what Bash does
- Large collections of completion recipes exist (home brew is your friend)

Living on the command-line

- `cd -` - go back to previous folder
- `file` *`file`* - guess file content (magic numbers)
- `lsof` - list open files
- `ps` (`aux` or `-ef`) and `top` - show processes
- Simple watch:

```
while true ; do clear ; command ; sleep
n ; done
```

OS X specific commands

- `open` *`file`* - Starts registered program and open file
- `say “Hello world”` - speech synthesis (download extra voices/languages in System preferences)
- `ls | pbcopy` - copy stdin to paste board
- `pbpaste` - paste to stdout
- `dns-sd -B _ssh._tcp` - show Bonjour enabled SSH hosts

Useful utilities

Find files: `find . -name ‘*.o’ -delete`

Patterns: `grep -r list *`

Cut field: `cut -f1,3 -d: /etc/passwd`

Word count: `wc -l *.cpp`

Transform: `tr “ “ “_” < README.org`

Sort lines: `sort -t: -n -r -k 4 /etc/passwd`

Last lines: `tail /etc/passwd`

First lines: `head /etc/passwd`

sed - the stream editor

joinash, https://www.flickr.com/photos/joinash/

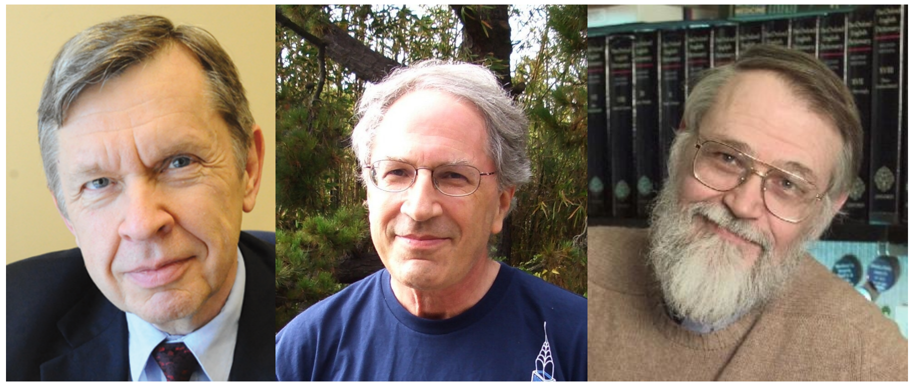
- sed is used to edit files non-interactively
- Option `-E` gives an editing (regular) expression
  - `s/FISH/HORSE/g` - substitute
  - `/FISH/d` - delete lines

Option `-i` is tricky:

- GNU sed has optional extension
- BSD sed requires extension (`''` is useful)

awk - a processing tool

- awk is a programming language by itself
- Matching lines are processed
    - line is split in fields (spaces are default)


A. Aho, P. Weinberger, B. Kernighan

Patterns:
`BEGIN` - before opening file
`END` - after closing file

```
cat foo | awk 'BEGIN {total=0 } END { print total } { total+=$1 }'
```

Scripting

Bash for programmers

- Bash is a complete programming language
  - Shell scripts grow and become ugly 😞
- Execution:
  - `sh script.sh`
  - `chmod +x script.sh; ./script.sh`
- Interpreted language → slow

Basic syntax

- White spaces: space and tab
- Comments: `#` and to end-of-line
- Statements: either end-of-line of `;` (semicolon)
- Variables and functions: Letters, digits and underscore

```
#!/bin/bash
# Monte Carlo calculation of pi
NSTEPS=500
NHITS=0
i=0
while [ $i -lt $NSTEPS ]; do
    x=$(echo $RANDOM/32767 | bc -l)
    y=$(echo $RANDOM/32767 | bc -l)
    d=$(echo "sqrt($x*$x+$y*$y) < 1.0" | bc -l)
    if [ $d -eq 1 ]; then
        NHITS=$(($NHITS + 1))
    fi
    i=$(($i + 1))
done

PI=$(echo "4.0*$NHITS/$NSTEPS" | bc -l)
echo "PI = $PI"
```

Variables

- Case-sensitive names
- No declarations, no types
- Strings: `"…"` are substituted; `'…'` are not
- Assignment (=): no spaces!
  - `$(…)` assignment from stdout including spaces
  - I often use `awk '{print $1}'` to remove spaces
  - `$((…))` arithmetic
- `$`*`varname`* - value of variable *`varname`*

Built-in variables:

- `$#` is the number of argument
- `$1`, `$2`, … are the arguments
- `$$` is the process ID
- `$?` is exit code of last command

```
#!/bin/bash
message_1="Hello"
message_2="World"
message="$message_1 $message_2” # concatenate
echo $message

# assign with output of command
nusers=$(grep -v ^# /etc/passwd | wc -l | awk
'{print $1}')
echo "Number of users: $nusers"

# do the math
answer=$((6*7))
echo "The life, the universe, and everything:
$answer"
```

Branches

- Simple branching with `if then else fi`
  - Enclose condition with `[]`
  - `elif` is possible, too
- Use `case in esac` when you can many cases and single condition

String operators:
`-z` is empty?
`-d` is directory?
`-f` is file?
`==` equal to
`!=` not equal to

Integer operators:
`-eq` equal to
`-lt` less than
`-ne` not equal to
`-gt` greater than

branches.sh

```
#!/bin/bash

if [ -z "$1" ]; then
    name="Arthur"
else
    name="$1"
fi

if [ "$name" != "Arthur" ]; then
    echo "Not Arthur"
else
    echo "Hello Arthur"
fi

answer=$((6*7))
if [ $answer -eq 42 ]; then
    echo "Life, the universe, and everything"
fi
```

branches.sh - con’t

```
 case "$name" in
     "Arthur")
       echo "Welcome onboard"
       ;;
     "Trillian")
       echo "You know Arthur"
       ;;
     *)
       echo "Who are you?"
       ;;
esac
```

Loops

- Simple loops: `for … in … ; do … done`
    - The seq utility can generate list of numbers
- Conditional loops: `while … ; do … done`
- Line-by-line: `while read line ; do … done`

One-liner (similar to `watch`)

```
while [ true ]; do
  clear;
  echo $RANDOM;
  sleep 1;
done
```

loops.sh

```
#!/bin/bash

# Multiplication table
for i in $(seq 1 10); do
    echo "$i $((3*$i))"
done

# All .sh files
for f in $(ls *.sh); do
    echo "$f $(head -1 $f | cut -c3-) $(wc -l $f | awk '{print $1}')"
done

# read self line-by-line
i=1
cat $0 | while read line ; do
    nchars=$(echo "$line" | wc -c | awk '{print $1}')
    echo "$i $nchars"
    i=$(($i+1))
done | sort -n -k 2
```

Functions

- Functions can increase readability of your scripts
- arguments are `$1`, `$2`, …
- `local` variables can be used
- `return` an integer and get it as `$?`
- Use global variable to return a string 😒

function.sh

```
#!/bin/bash

mult () {
    local n=$1
    return $((3*$n))
}


for n in $(seq 1 10); do
    mult $n
    echo "$n $?”
done
```

Tips and tricks

- Use `set -e` to exit early
    - or use `|| exit 1`
- `set -O pipefail` and you can get the exit code of the first failing program in a pipe
    - `xcpretty` never fails but `xcodebuild` might
- Use `tee` to write to stdout and file
- To trace (debugging): `set -x` or `sh -x`

Tips and tricks

- Always use `“$var”` when dealing with file names (and strings)
  - `str=“fish horse”; for i in $str; do echo $i; done`
  - `str=“fish horse”; for i in “$str”; do echo $i; done`
- Call `mkdir -p` when creating folders
- Create temp. files with `mktemp /tmp/$$.XXXXXX`
- Using variable to modify behaviour of script:
  - `FLAGS=“-O3 -libc++=stdlibc++” build.sh`
- Subshells: `(cd foo && rm -f bar)`

Tool for developers

Home brew

- Home brew provides calories for console cowboys
- You don't have to be *root* to install
- Software is installed in `/usr/local/Cellar`, and symlinked to `/usr/local/bin`
- Brew cask is for binary distribution
- http://brew.sh and http://caskroom.io

Greg Peverill-Conti, https://www.flickr.com/photos/gregpc/

Examples:
```
brew search bash
brew info bash
brew install bash
brew update
```

Tools for developers

- Apple provides some basic tools
  - `nm` - display symbol table
  - `c++filt` - Prettify C++ and Java names
  - `otool -L` - display which shared libraries are required
  - `libtool` - create libraries
  - `lipo` - manipulate fat/universal binaries

zzpza, https://www.flickr.com/photos/zzpza/

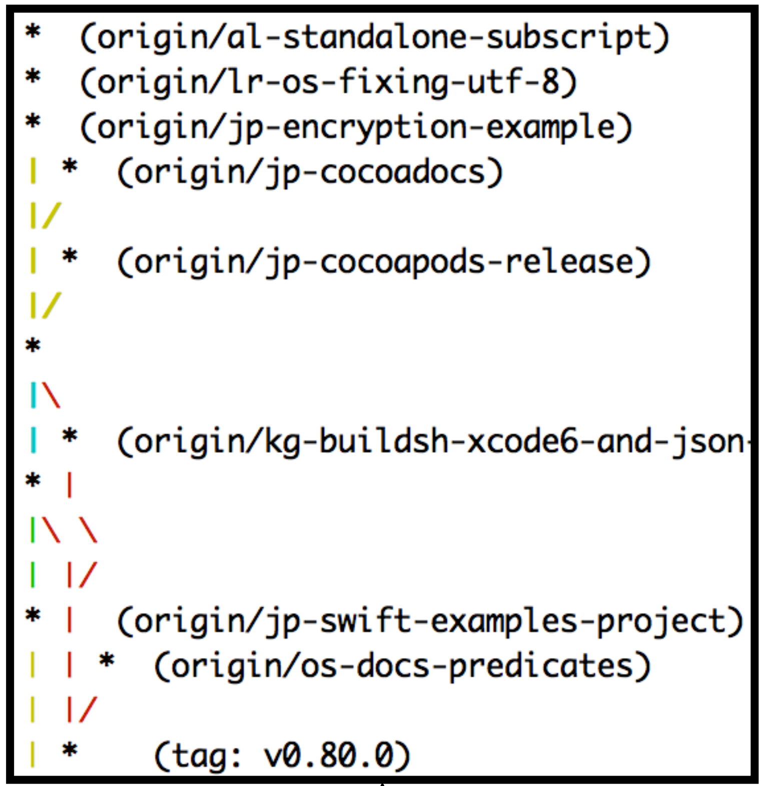
Examples:

```
nm book.o | c++filt
otool -L RealmInspector
```

git

- Home brew packages:
  - git, git-extras
  - Symlink `/usr/local/bin/git` to `/usr/bin`
- Bash completion works
  - commands, branches, etc.
- Fancy prompt:

```
PS1='\u@\h:\w$(__git_ps1 " (%s)") \$ '
```



```
*   (origin/al-standalone-subscript)
*   (origin/lr-os-fixing-utf-8)
*   (origin/jp-encryption-example)
| *   (origin/jp-cocoadocs)
|/
| *   (origin/jp-cocoapods-release)
|/
*
|\
| *   (origin/kg-buildsh-xcode6-and-json
* |
|\ \
| |/
* |   (origin/jp-swift-examples-project)
| | *   (origin/os-docs-predicates)
| |/
| *     (tag: v0.80.0)
```

Examples:
```
git count -all
git contrib "Kenneth Geisshirt"
```

```
git log --graph --simplify-by-decoration --pretty=format:'%d' --all
```

Xcode

- You can build Xcode projects at the command-line

```
xcodebuild -scheme OreDevPlain -
configuration Release -sdk
iphonesimulator
```

- Targets: clean, build, test
- You can add shell scripts to build phases

xcpretty

- The output of xcodebuild can be hard to read
- xcpretty makes it prettier
- Installation:
  - `sudo gem install xcpretty`
- Usage:
  - `xbuildcode … | xcpretty`

xctool

- Yet another build helper
- Installation:
    - `brew install xctool`
- Usage:
    - `xctool -scheme OredevPlain -configuration Release -sdk iphonesimulator build`

Further information

- *Classical Shell Scripting*. R. Arnolds and N.H.F. Beebe. O'Reilly Media, 2005.
- The sed FAQ: http://sed.sourceforge.net/sedfaq.html
- Advanced Bash-Scripting Guide: http://www.tldp.org/LDP/abs/html/

http://realm.io
We are hiring