

Moving to zsh



Scripting OS X

Armin Briegel

Mac Admin, Consultant and Author

pro warehouse

Practical Scripting

 **WWDC19**

San Jose, CA, June 3-7



Use zsh as the default shell on your Mac

Starting with the macOS Catalina beta, your Mac uses zsh as the default login shell and interactive shell. You can make zsh the default in earlier versions of macOS as well.

By default, your Mac uses either zsh or bash as the command-line interpreter for the login shell and interactive shell:

- **zsh** (Z shell) is the default shell for all newly created user accounts, starting with the macOS Catalina beta, currently available only to members of the Apple Developer Program.
- **bash** is the default shell in [macOS Mojave and earlier](#).

zsh is highly compatible with the Bourne shell (sh) and mostly compatible with bash, with some differences. For more about zsh and its comprehensive command-line completion system, enter `man zsh` in Terminal.

How to change your default shell

Practical Scripting

Moving to zsh



Moving to zsh

Apple has announced that in macOS 10.15 Catalina the default shell will be zsh.

In this series, I will document my experiences moving bash settings, configurations, and scripts over to zsh.



Moving to zsh

Apple has announced that in macOS 10.15 Catalina the default shell will be zsh.

In this series, I will document my configuration, and scripts over to zsh.



Moving to zsh, part 2: Configuration

Apple has announced that in macOS 10.15 Catalina the default shell will be zsh.



Moving to zsh, part 3: Shell Options

Apple has announced that in macOS 10.15 Catalina the default shell will be zsh.



Moving to zsh



Moving to zsh, part 5:



Moving to zsh, part 2: Configuration Files



Moving to zsh, part 6 – Customizing the zsh Prompt



Moving to zsh, part 3: Shell Options



Moving to zsh – part 7: Miscellanea



Moving to zsh, part 4: A



Moving to zsh, part 8 –

Shellcheck and zsh

The sad news is that the `shellcheck` binary does not really know how to deal with zsh scripts:

```
% shellcheck script.zsh
```

Get Current User in Shell Scripts on macOS

...or, how to deal with deprecated

Make zsh show working directory in Terminal window title in macOS

While I was working on customizing my zsh configuration files for my zsh article series, I noticed that Terminal would not display the current working directory when using zsh. It

scriptingosx.com/zsh



Use zsh as the default shell on your Mac

Starting with the macOS Catalina beta, your Mac uses zsh as the default login shell and interactive shell. You can make zsh the default in earlier versions of macOS as well.

By default, your Mac uses either zsh or bash as the command-line interpreter for the login shell and interactive shell:

- **zsh** (Z shell) is the default shell for all newly created user accounts, starting with the macOS Catalina beta, currently available only to members of the Apple Developer Program.
- **bash** is the default shell in [macOS Mojave and earlier](#).

zsh is highly compatible with the Bourne shell (sh) and mostly compatible with bash, with some differences. For more about zsh and its comprehensive command-line completion system, enter `man zsh` in Terminal.

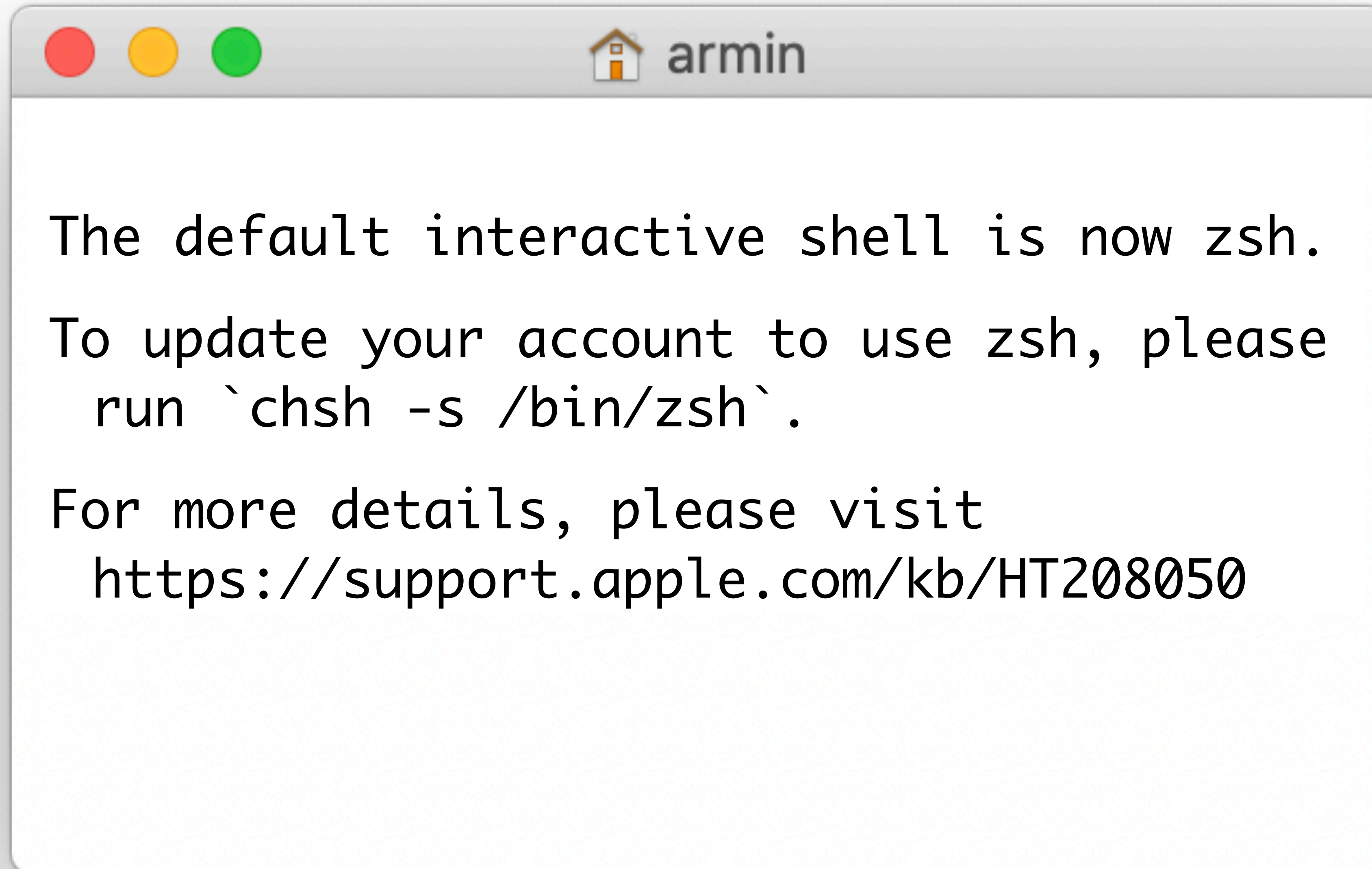
How to change your default shell

Changes in Catalina

Default shell for *new* users is now `/bin/zsh`

Existing users keep their shell

bash in Catalina

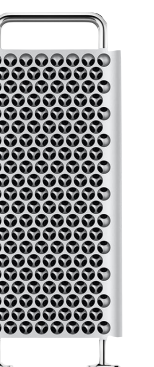


(Artist Impression, not a real screenshot)

History of shells (simplified)



1970 1980 1990 2000 2010 now





10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9 10.10 10.11 10.12 10.13 10.14 10.15



//biim//boosh



PANIC

Don't Panic! (yet)

/bin/bash still in Catalina 🌴

Many scripts depend on `/bin/bash`

Management, Installation, Workflow, Applications

Apple, 3rd party, MacAdmins



Apple is messaging to move away from `/bin/bash`

When?



zsh

"zee-shell" 

Shell Names

sh

Thompson/Posix shell


bsh

Bourne shell

bash

Bourne again shell



zsh 

zee-shell



zsh 

zed-shell



Shell Namespace

```
% echo {a..z}{,a,e,i,o,u}sh
```

ash	dsh	gsh	jsh	msh	psh	ssh	vsh	ysh
aash	dash	gash	jash	mash	pash	sash	vash	yash
aesh	desh	gesh	jesh	mesh	pesh	sesh	vesh	yesh
aish	dish	gish	jish	mish	pish	sinh	vish	yish
aosh	dosh	gosh	josh	mosh	posh	sosh	vosh	yosh
aush	dush	gush	jush	mush	push	sush	vush	yush
bsh	esh	hsh	ksh	nsh	qsh	tsh	wsh	zsh
bash	eash	hash	kash	nash	qash	tash	wash	zash
besh	eesh	hesh	kesh	nesh	qesh	tesh	wesh	zesh
bish	eish	hish	kish	nish	qish	tish	wish	zish
bosh	eosh	hosh	kosh	nosh	qosh	tosh	wosh	zosh
bush	eush	hush	kush	nush	qush	tush	wush	zush
cs	fsh	ish	lsh	osh	rsh	ush	xsh	
cash	fash	ias	lash	oash	rash	uash	xash	
cesh	fesh	iesh	lesh	oesh	resh	uesh	xesh	
cish	fish	iish	lish	oish	rish	uish	xish	
cosh	fosh	iosh	losh	oosh	rosh	uosh	xosh	
cush	fush	iush	lush	oush	rush	uush	xush	

Shell Namespace


 sh

 sh

 sh


 sh

 sh

 sh

 sh

 sh


 sh


 sh

 sh

 sh

 sh

 sh

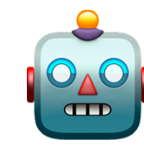
 sh


 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh

 sh


 sh

 sh

 sh

 sh

 sh

 sh

 sh



iShell

sh Pro

Prosh

sh+

*subscription required

Xsh

Ten-shell ?

Ecks-shell ?

Ten-Ess-Aitch ?

Xish ?

Xsh

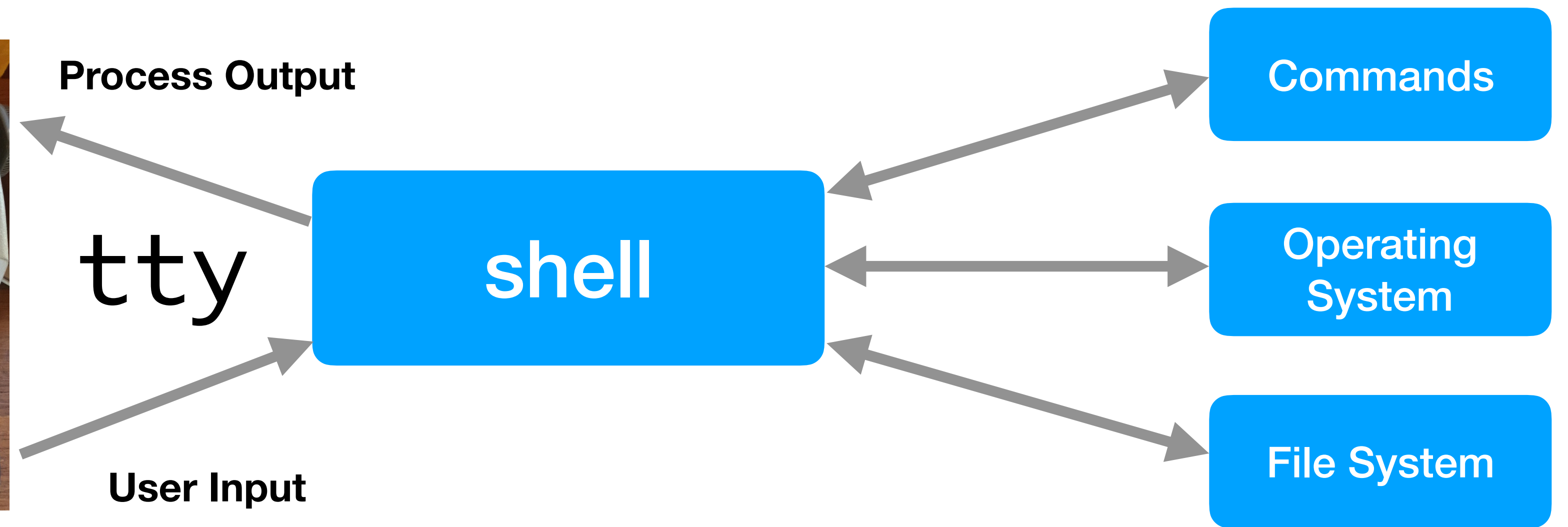
zsh

"zee-shell" 

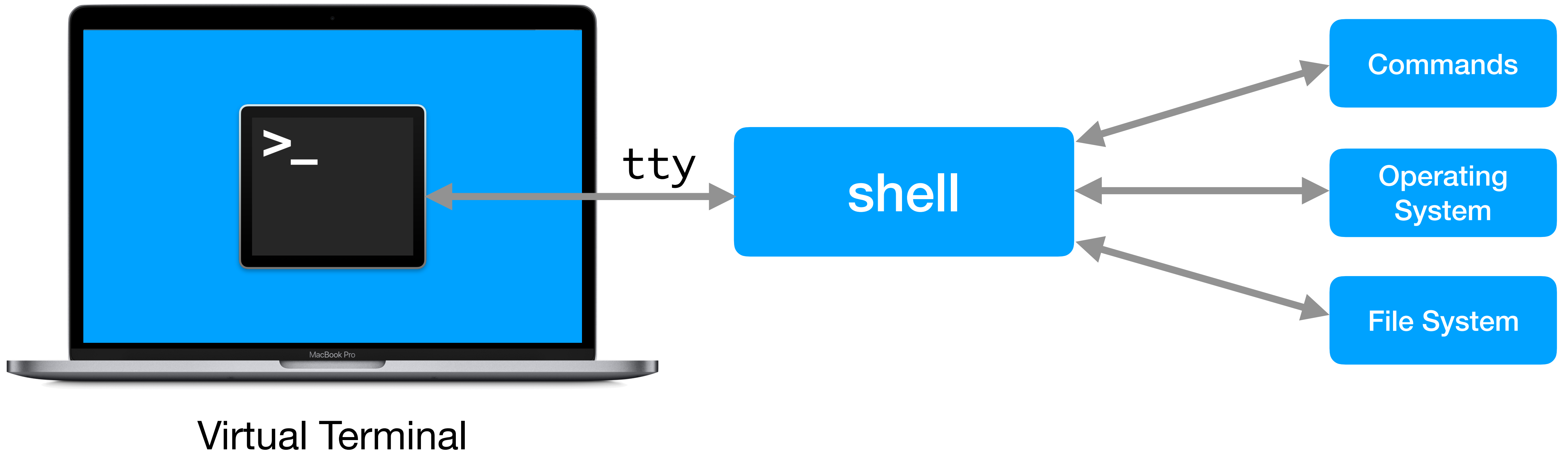
What is a shell?



Terminal



What is a shell?





Interactive Terminal
UserShell: /bin/zsh



Interpret Script Files
#!/bin/sh



Interactive Terminal
UserShell: /bin/zsh



Interpret Script Files
#!/bin/sh

zsh

"zee-shell" 

zsh Features

Compatibility modes

Loadable Functions/Modules

Programmable completion

Named directories

Shared command history

where command

Extended Globbing

Auto cd

Auto Correction

MultiIO

Prompt Themes

etc.





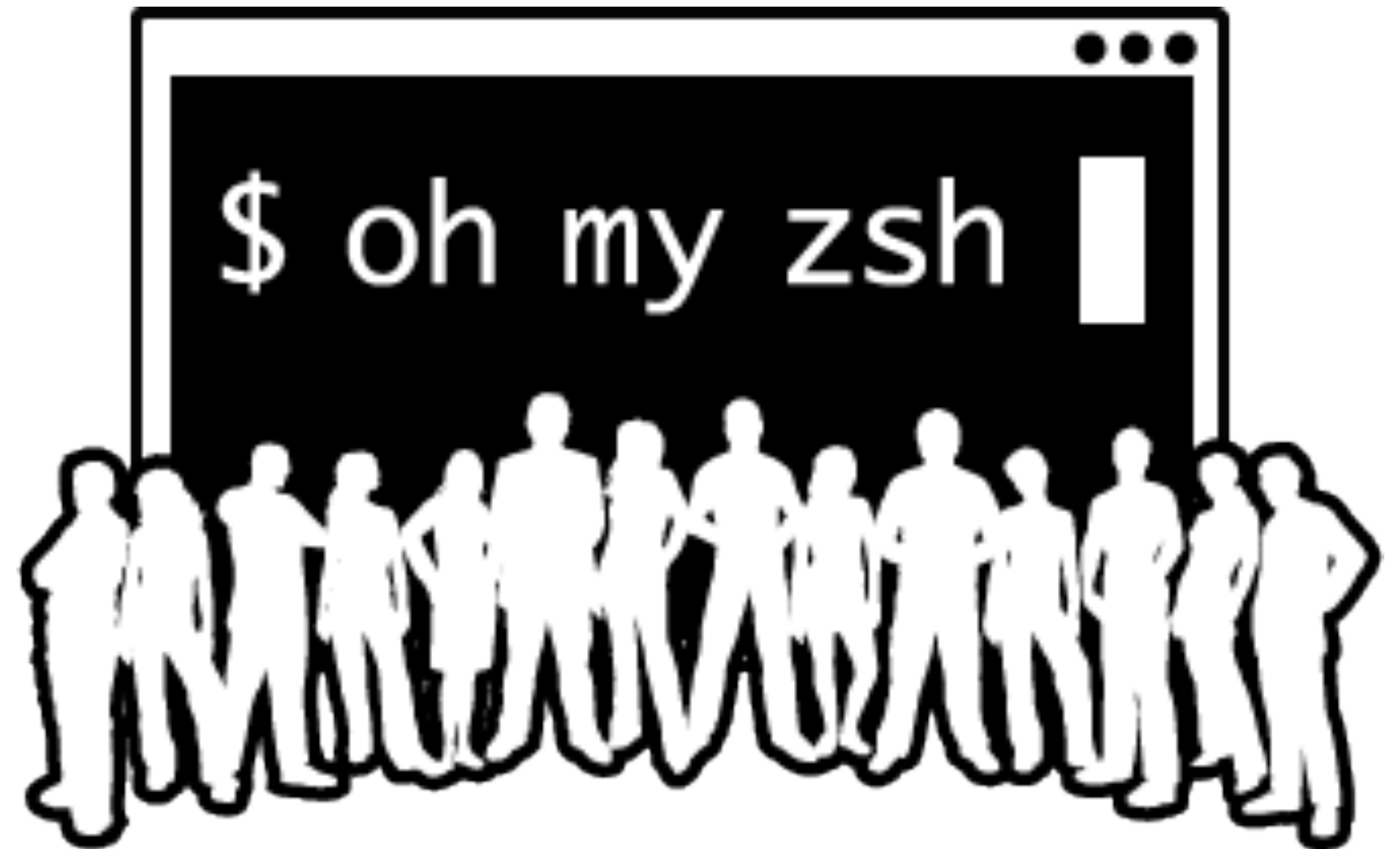
Modules and Themes

oh my zsh

prezto

antigen

Powerlevel9k/10k



Lean Style

~/powerlevel10k master ↑2 > | ✨ minikube

~/powerlevel10k on 🐱 🍷 master ↑2 at ✨ minikube
> |

Classic Style

~/powerlevel10k > master ↑2 | minikube ✨

~/powerlevel10k / master ↑2 minikube ✨

~/powerlevel10k / 🐱 🍷 master ↑2 minikube ✨
> |

Switch to zsh

Switch to zsh

zsh since Cheetah 🐆

zsh v5 since Mavericks 🌊

zsh 5.3 in High Sierra 🏔️ and Mojave 🌄

zsh 5.7.1 in Catalina 🌴

Directory Utility

Services Search Policy **Directory Editor**

Viewing Users in node /Local/Default Not authenticated

Search: armin

Armin Briegel

Name	Value
PrimaryGroupID	20
RealName	Armin Briegel
▶ RecordName	armin
RecordType	dsRecTypeStandard:Users
UniqueID	501
UserShell	/bin/zsh
dsAttrTypeNative:_writers_...	armin
dsAttrTypeNative:_writers_...	armin
dsAttrTypeNative:_writers_j...	armin
dsAttrTypeNative:_writers_...	armin


+ - Text Data

/bin/zsh

+ - 1 record Revert Save


Users & Groups Search


Current User

 **Armin Briegel**
Admin

Advanced Options...

Other Users

 **Guest User**
Off

 **Login Options**

Armin Briegel


Password **Login Items**

Change Password...

Contacts Card: Open...

Allow user to administer this computer

Enable parental controls **Open Parental Controls...**

 Click the lock to prevent further changes. ?

Users & Groups Q Search

Advanced Options

User: "Armin Briegel"

WARNING: Changing these settings might damage this account and prevent the user from logging in. You must restart the computer for the changes to these settings to take effect.

User ID:

Group:

Account name:

Full name:

Login shell:

Home directory:

UUID:

Apple ID:

Aliases:

/bin/tcsh

/bin/sh

/bin/csh

/bin/zsh

/bin/ksh

+ -



armin

```
$ chsh -s /bin/zsh
```



armin

```
[$ chsh -s /bin/zsh ]
```

Changing shell for armin.

Password for armin: 



armin

MacBook%



Change Default Shell

```
chsh -s /bin/zsh username
```

```
dscl . change /Users/username /bin/bash /bin/zsh
```

What happens to bash scripts?

Script interpreter is determined by shebang

Interactive shell and script interpreter are independent

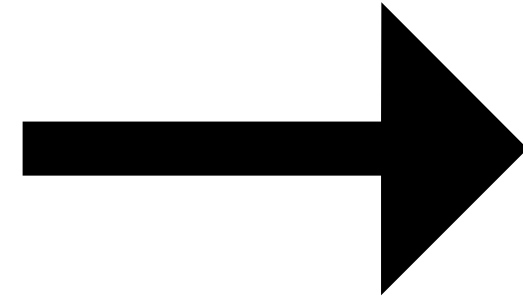
Use a different interactive shell and script interpreter

No shebang: use current shell as interpreter

General usage stays the same

Some key strokes are slightly different

\$



%

zsh Features

Multi IO

Redirect streams to multiple targets

```
system_profiler SPHardwareDataType >file1.txt >file2.txt
```

Multi IO

Redirect streams to multiple targets

```
system_profiler SPHardwareDataType >file1.txt | cat
```

Multi IO

Redirect streams to multiple targets

```
system_profiler SPHardwareDataType >hardwareprofile.txt  
| awk '/Serial Number/ { print $4 }' >&1 >serial.txt
```

Multi IO

Multiple input streams are concatenated

```
sort </usr/share/calendar/calendar.freebsd  
    </usr/share/calendar/calendar.computer
```

Redirection without command

```
% < file.txt
```

Equivalent to:

```
% more file.txt
```


Auto cd

When you enter a directory path, zsh assumes cd

```
% Documents
```

```
% pwd
```

```
/Users/armin/Documents
```

Enable with: `setopt autocd`

Correction

```
% daet
```

```
zsh: correct 'daet' to 'date' [nyae]? y
```

```
Thu Aug 22 14:15:00 CEST 2019
```

Enable with: `setopt correct`

Shell Options

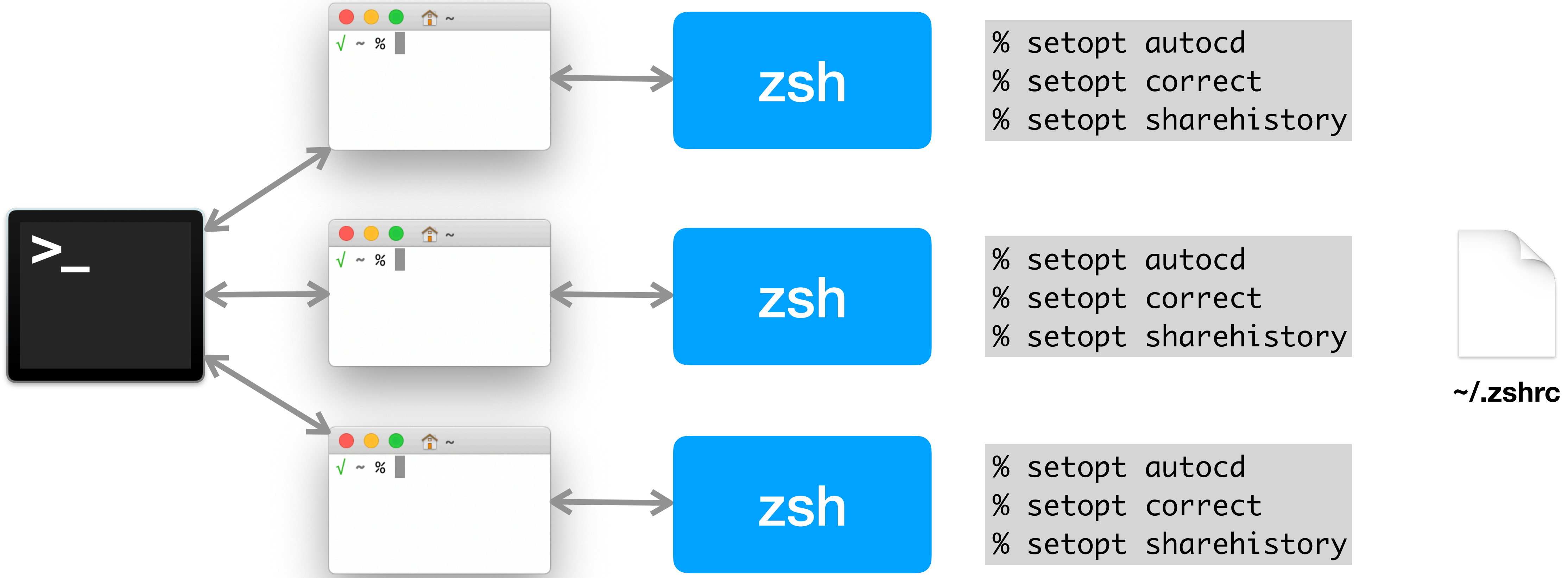
```
setopt autocd
```

```
setopt sharehistory
```

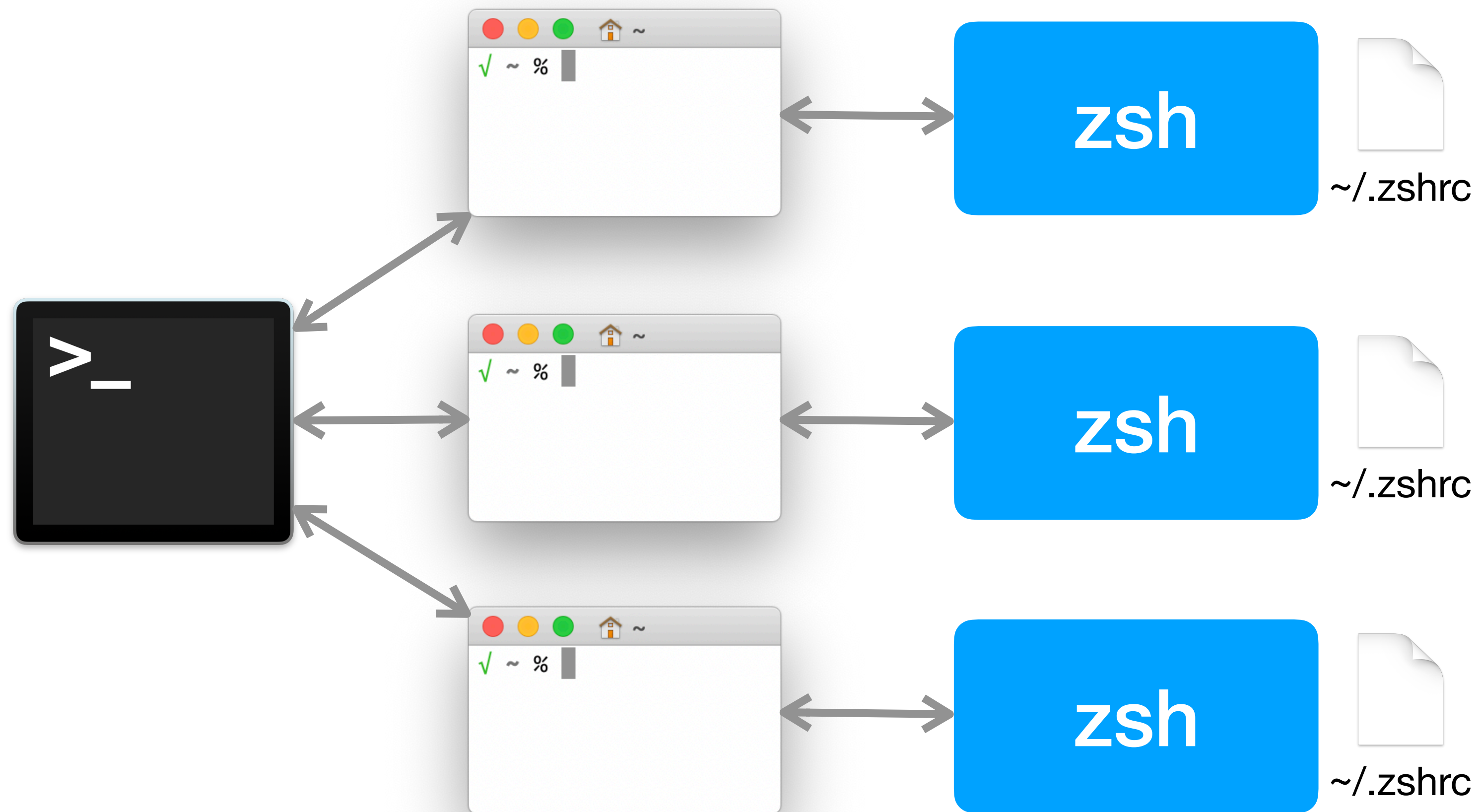
```
setopt correct
```

Many more... (RTM)

Each new window has a new shell



Each new window has a new shell



Configuration Files

Configuration Files

All users	User	Login shell	Interactive	Scripts	Terminal
/etc/zshenv	→ ~/.zshenv	✓	✓	✓	✓
/etc/zshprofile	→ ~/.zshprofile	✓			✓
/etc/zshrc	→ ~/.zshrc	✓	✓		✓
/etc/zshlogin	→ ~/.zshlogin	✓			✓

User Configuration Files

Use `~/ .zshrc`

Put all *your* configurations in `.zshrc`

Transfer info from `.bash_profile` or `.bashrc`

Tools, such as `prezto` or `oh-my-zsh`:
→ read documentation to avoid conflicts

Consider order of files

Managing Central Config Files

`/etc/zlogin`

overrides user settings in `~/ .zshrc`

Add a line to `/etc/zshrc`, which sources your config file

```
[[ -r /etc/zshrc_myorg ]] && source /etc/zshrc_myorg
```

redo after updates

Custom Prompt

Default zsh prompt

Stored in PS1, PROMPT or prompt

≤ Mojave 🌅

```
MacBook%
```

```
prompt=' %m%# '
```

Catalina 🌴

```
armin@MacBook Documents %
```

```
prompt=' %n@%m %1~ %# '
```

Customize your prompt

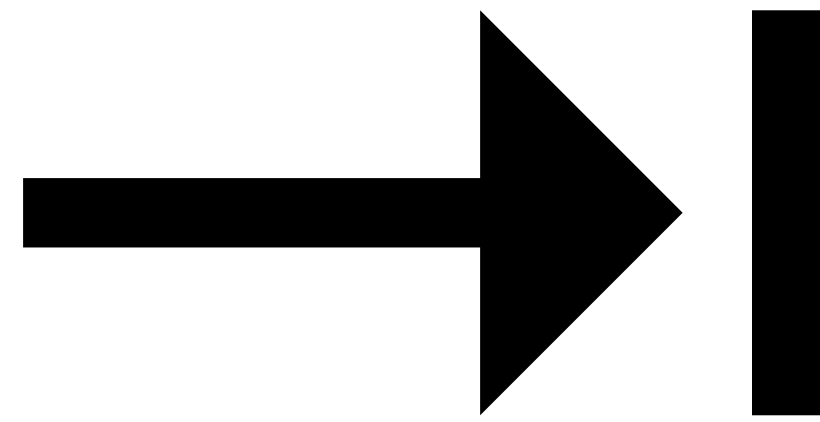
```
PROMPT= %(? .%{😊%} .%{👉%} ) %F{blue}%2~%f %(! .#.→)
```



~/Documents



Completions



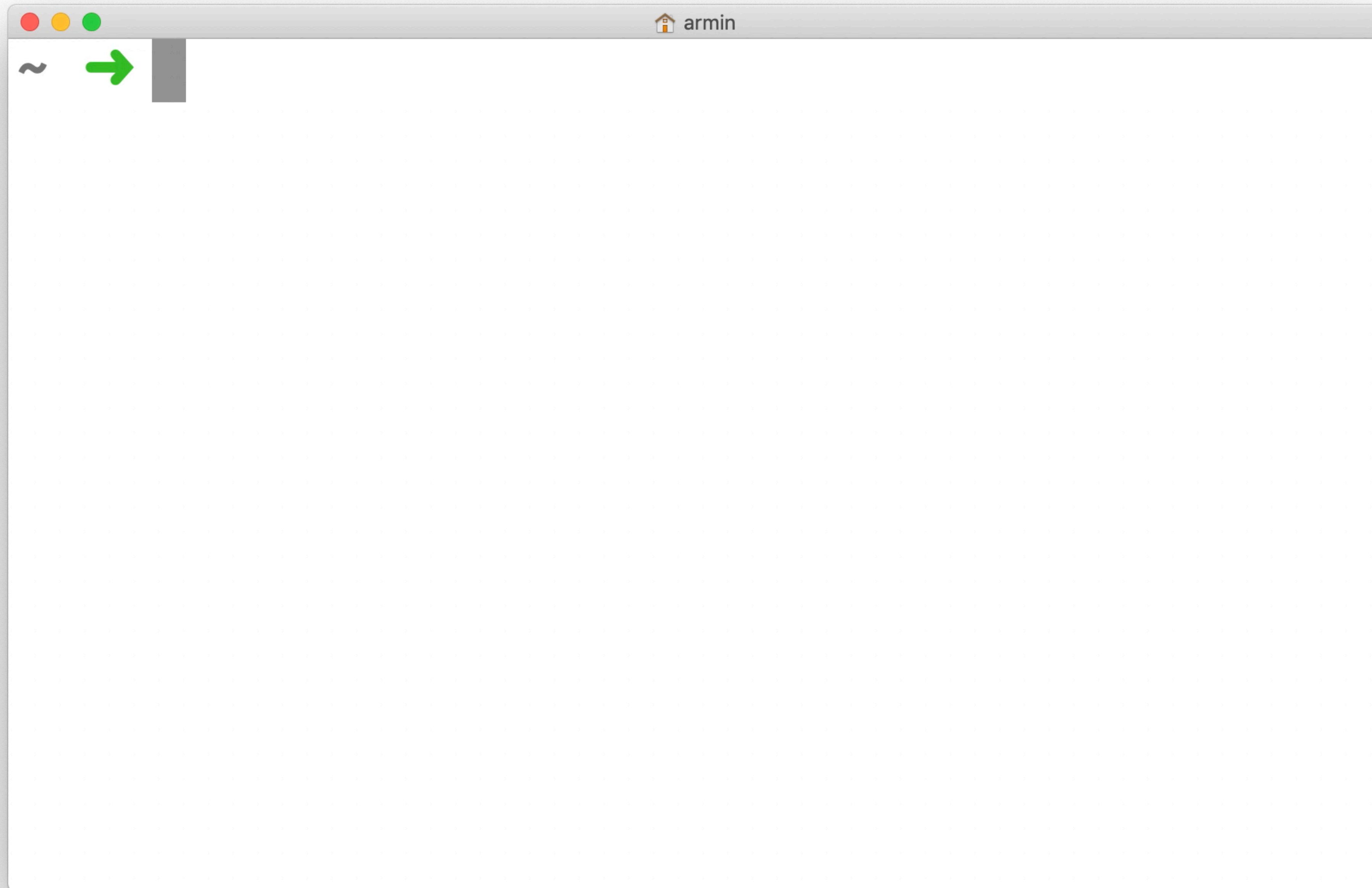
Tab-completion

Default zsh tab completion: commands and paths

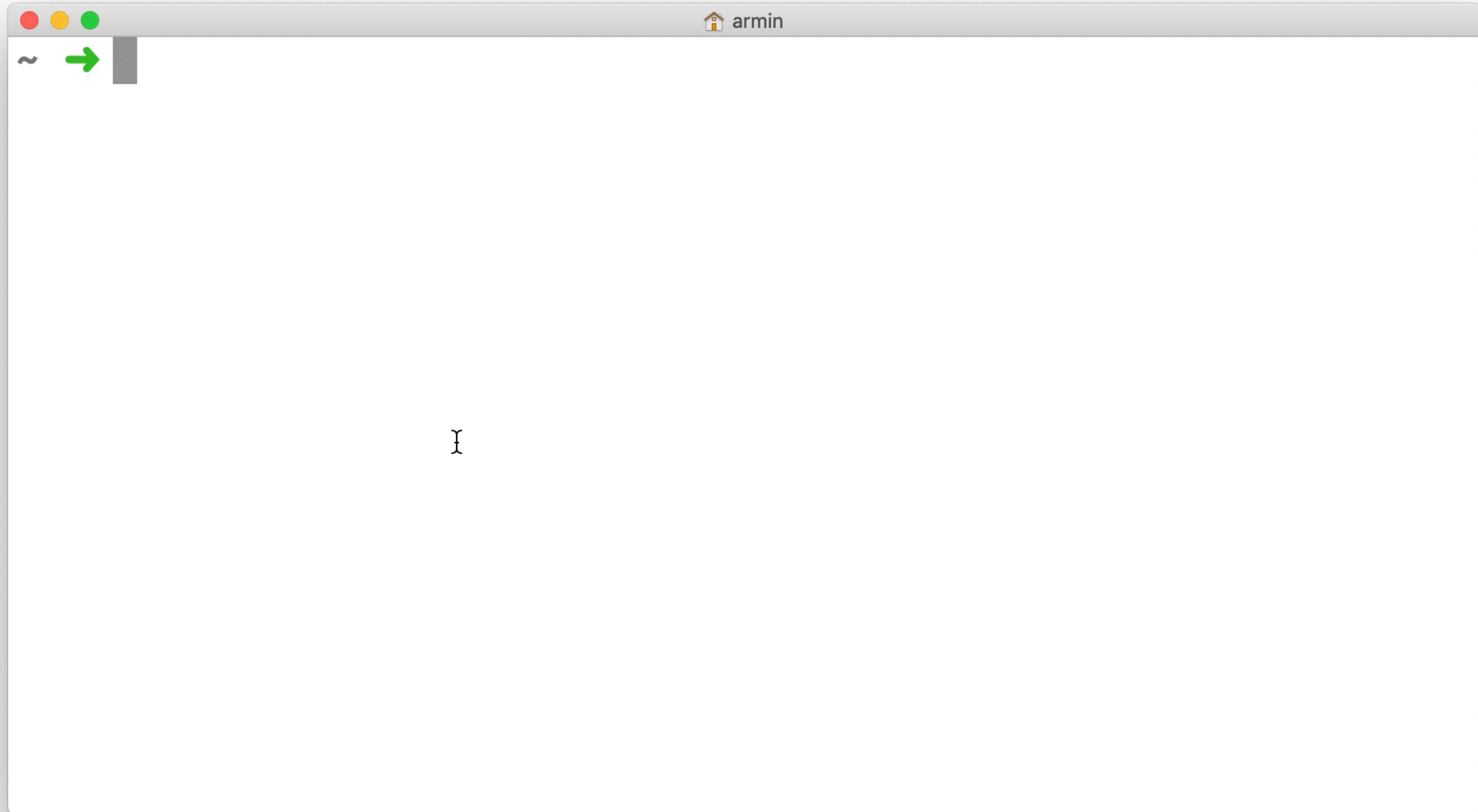
Load `compinit` for more:

```
autoload -Uz compinit && compinit
```

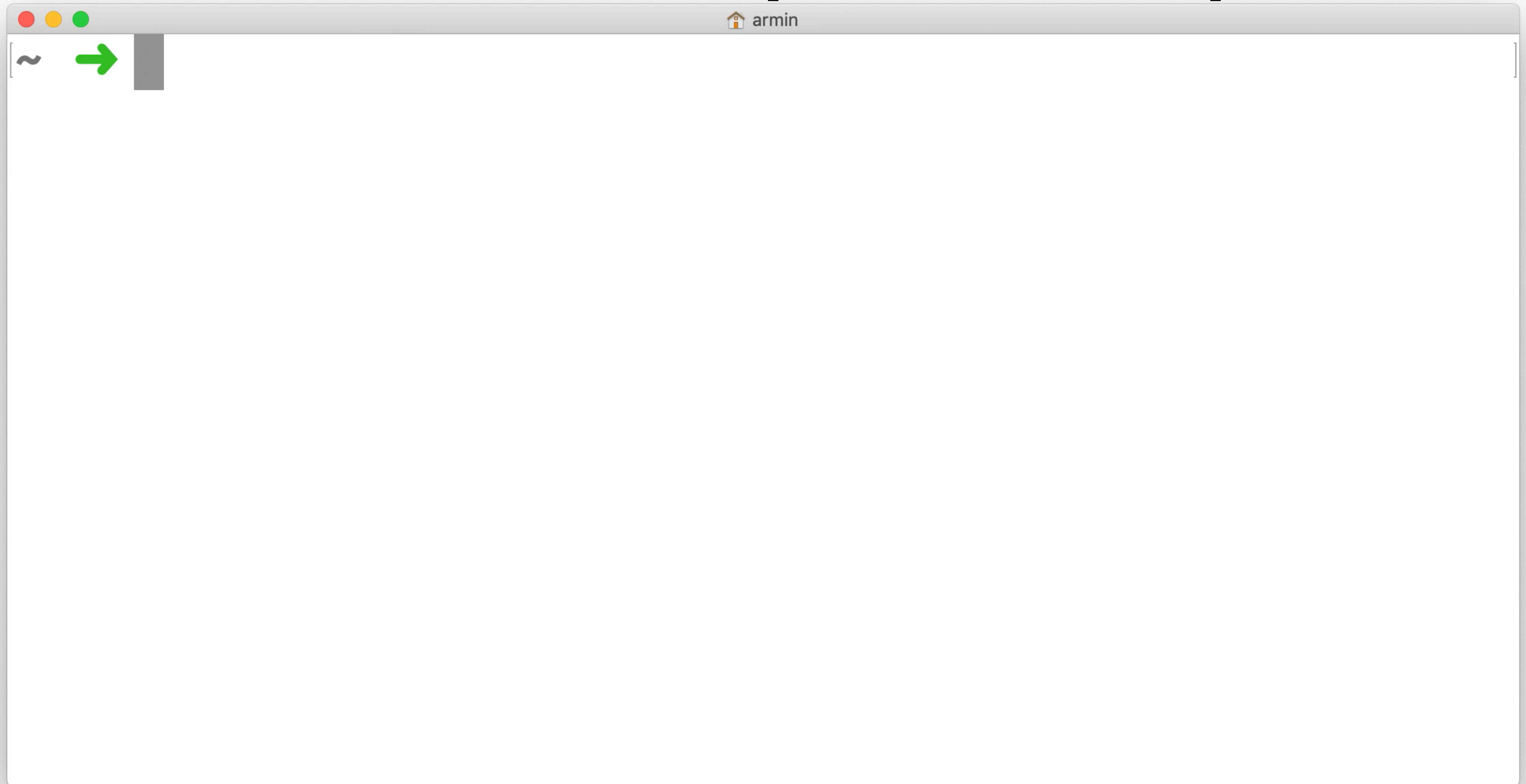
Partial Completions



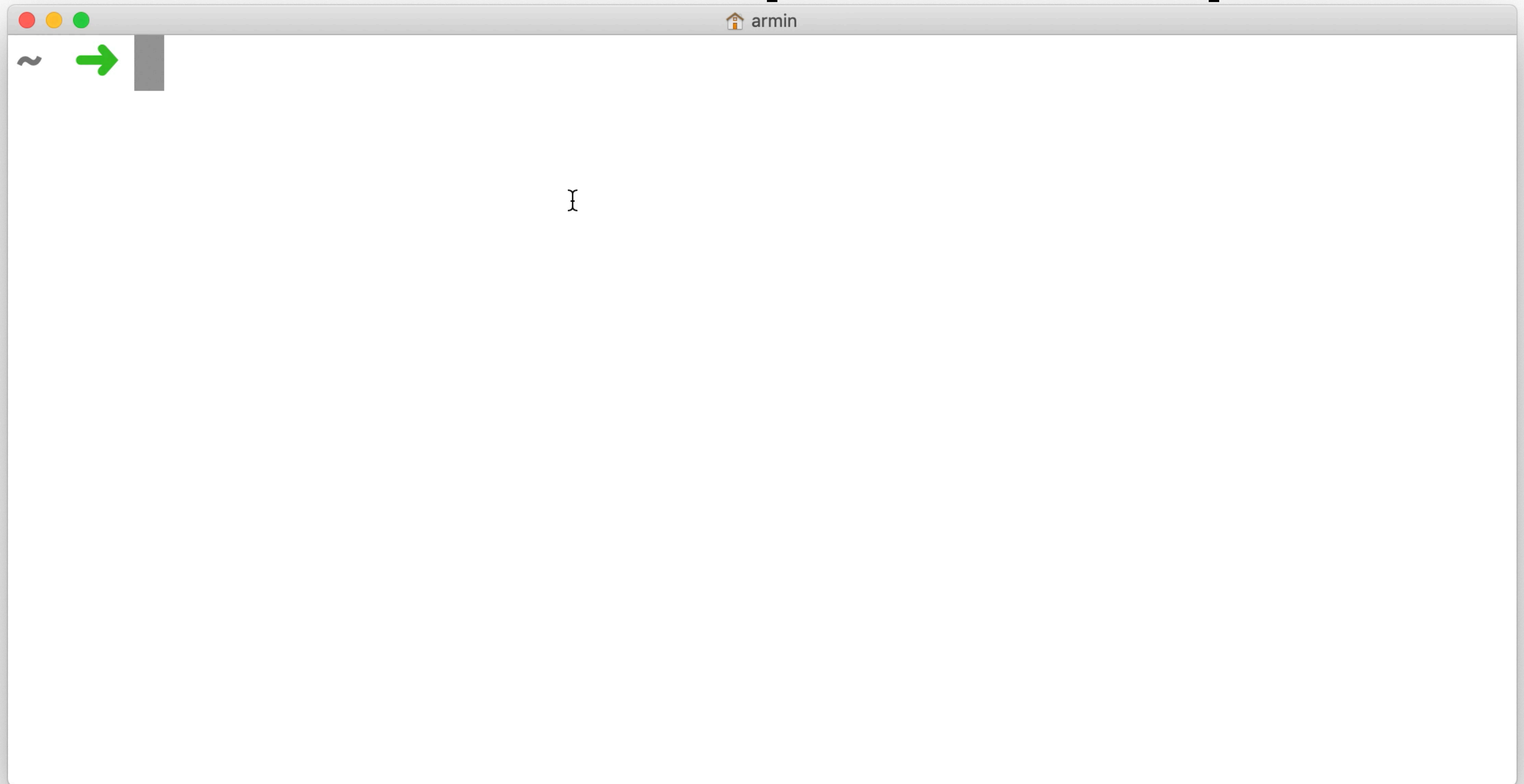
Argument Completion



Command and Option Completion



Command and Option Completion



Mac Specific Completion Rules

High Sierra /Mojave  (zsh 5.3):

defaults

fink

hdiutil

open

softwareupdate

system_profiler

Mac Specific Completion Rules

Catalina 🌴 (zsh 5.7.1):

caffeinate

defaults

fink

fs_usage

hdiutil

mdfind

mdls

mdutil

networksetup

nvrnm

open

osascript

otool

pbcopy/pbpaste

plutil

say

sc_usage

scselect

scutil

softwareupdate

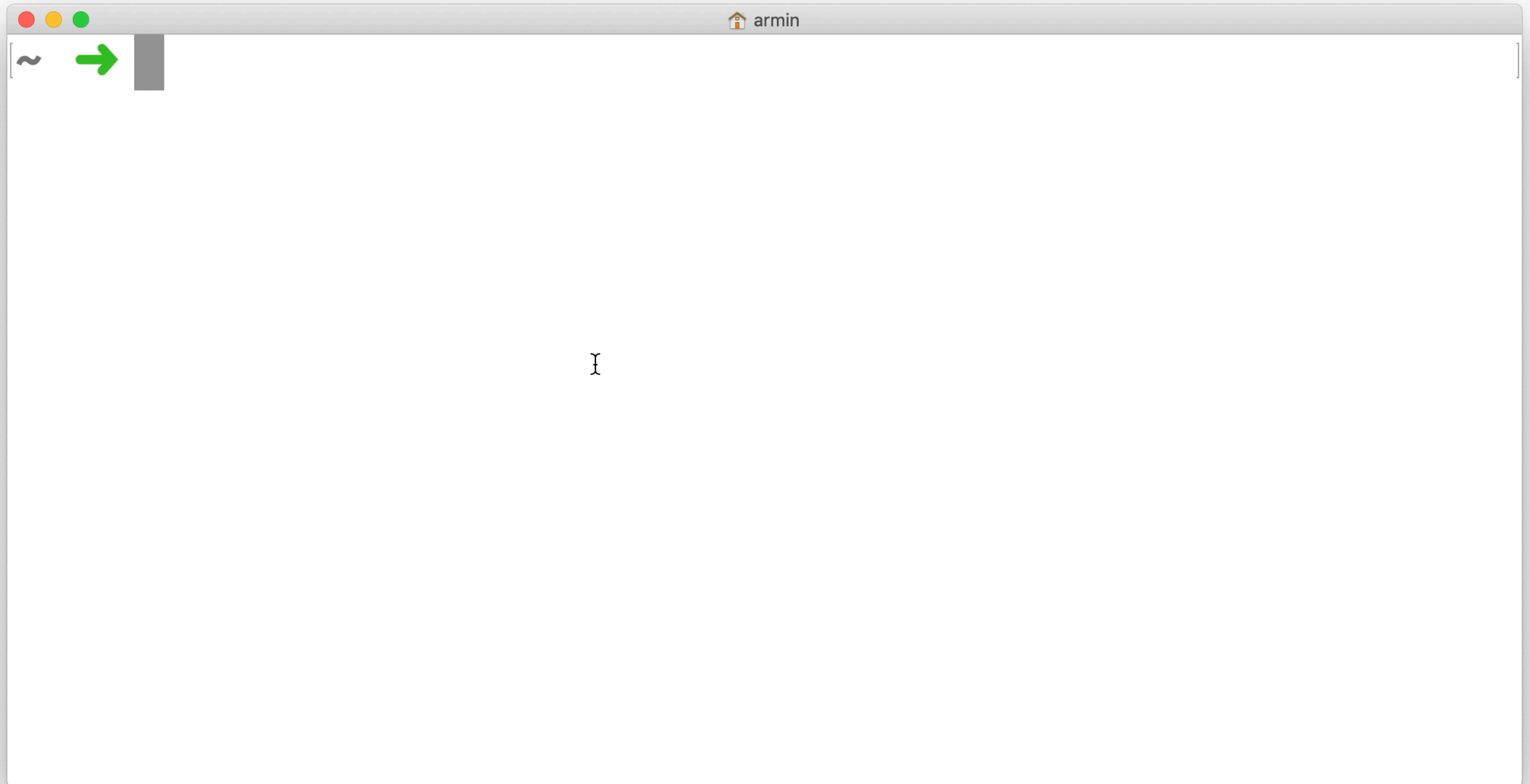
sw_vers

swift

system_profiler

xcode_select

Command and Option Completion



Build your own!

github.com/scriptingosx/mac-zsh-completions

desktoppr

xattr

sysadminctl

swift

dsccl

dseditgroup

bbedit

Contribute!

to e

tcutil

xed

launchctl

diskutil

installer

pkgbuild

fdsetup

jamf

productbuild

PlistBuddy



Interactive Terminal
UserShell: /bin/zsh



Interpret Script Files
#!/bin/sh

Scripting zsh

Replace the shebang

`#!/bin/bash` → `#!/bin/zsh`

Differences to bash scripting

Word splitting

Array indices

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}
```

```
wordlist="one two three four five"
```

```
countArguments $wordlist
```

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
countArguments one two three four five
```

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
countArguments one two three four five # -> 5
```

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
countArguments "$wordlist"
```

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
countArguments "one two three four five"
```


Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
countArguments "one two three four five" # -> 1
```

Recap: Word Splitting in sh/bash

```
function countArguments() {  
    echo "${#@}"  
}  
  
wordlist="one two three four five"  
  
countArguments $wordlist      # -> 5  
  
countArguments "$wordlist"    # -> 1
```

Word Splitting in **zsh**

```
function countArguments() {  
    echo "${#@}"  
}  
  
wordlist="one two three four five"  
  
countArguments $wordlist      # -> 1  
  
countArguments "$wordlist"    # -> 1
```

Word Splitting in zsh

```
function countArguments() {  
    echo "${#@}"  
}  
  
wordlist="one two three four five"  
  
countArguments ${=wordlist}
```

Word Splitting in zsh

```
function countArguments() {  
    echo "${#@}"  
}  
  
wordlist="one two three four five"  
  
countArguments ${=wordlist} # -> 5
```

Word Splitting in zsh

```
function countArguments() {  
    echo "${#@}"  
}  
  
wordlist="one two three four five"  
  
for x in ${=wordlist}; do  
    ...  
done
```

Word Splitting in zsh

```
function countArguments() {  
    echo "${#@}"  
}  
wordlist="one two three four five"  
  
for x in ( ${(s/ /)wordlist} ) do  
    ...  
done
```

Array Splitting

```
macOSversion=$(sw_vers -productBuild)    # 10.14.6  
versionList=${(s/./)macOSVersion}
```


Array Splitting

```
macOSversion=$(sw_vers -productBuild)      # 10.14.6  
versionList=${(s/./)macOSVersion}        # ( 10 14 6 )
```

Array Indices

```
macOSversion=$(sw_vers -productBuild)    # 10.14.6
versionList=${(s/./)macOSVersion}       # ( 10 14 6 )

echo ${versionList[1]}                  # -> 10
echo ${versionList[2]}                  # -> 14
echo ${versionList[3]}                  # -> 6
```

Array Indices

```
macOSversion=$(sw_vers -productBuild)      # 10.14.6  
versionList=${(s/./)macOSVersion}         # ( 10 14 6 )
```

```
setopt ksharrays
```

```
echo ${versionList[0]}                    # -> 10
```

```
echo ${versionList[1]}                    # -> 14
```

```
echo ${versionList[2]}                    # -> 6
```

Differences to bash scripting

Word splitting

Array indices

Scripting zsh

Scripting zsh?

Scripting zsh

zsh since Cheetah 🐆

zsh v5 since Mavericks 🌊

zsh 5.3 in High Sierra 🏔️ and Mojave 🌄

zsh 5.7.1 in Catalina 🌴

zsh scripts are backwards compatible

Scripting zsh



Exception: Recovery

Recovery

`/bin/bash`

~~`/bin/zsh`~~

`installr, bootstrapp, Twocanoes MDS`

`/bin/dash` in Catalina 🌴

dash?

dash

Debian Almquist Shell

Minimal implementation of POSIX sh

Stands in as sh in many Unix-like systems

Added to macOS Catalina 🌴

Recovery

bash as sh

```
% sh --version
```

```
GNU bash, version 3.2.57(1)-release (x86_64-apple-darwin18)
```

```
Copyright (C) 2007 Free Software Foundation, Inc.
```

Catalina 🌴: choose your sh

Symbolic link `/var/select/sh`

Determines which shell handles `/bin/sh`

`bash` (default), `zsh`, or `dash`

Future macOS

dash as sh

We don't know when... 🙄

bash as sh

```
#!/bin/sh

if [[ $(true) == $(true) ]]
then
    echo "still good"
else
    echo "nothing is true"
fi
```

```
% ./shtest.sh
still good
```

dash as sh

```
#!/bin/sh
```

```
if [[ $(true) == $(true) ]]
```

```
then
```

```
    echo "still good"
```

```
else
```

```
    echo "nothing is true"
```

```
fi
```

```
% sudo ln -sf /bin/dash /var/select/sh
```

```
% ./shtest.sh
```

```
./shtest.sh: 3 ./shtest.sh: [[: not found  
nothing is true
```

bash as sh

executes 'bashisms' without error

```
[[ ... ]] <<heredoc ≈
```

And has been doing so since 10.3

dash as sh will error on 'bashisms'

Check your sh scripts!

Check your sh scripts!

```
% shellcheck shtest.sh
```

```
In shtest.sh line 3:
```

```
if [[ $(true) == $(true) ]]
```

```
  ^-----^ SC2039: In POSIX sh, [[ ]] is undefined.
```

```
For more information:
```

```
shellcheck.net/wiki/SC2039 -- In POSIX sh, [[ ]] is undefined.
```

Check your sh scripts!

also check for Perl, Python, and Ruby one-liners

```
LoggedInUser=$(python -c 'from SystemConfiguration import
SCDynamicStoreCopyConsoleUser; import sys; username =
(SCDynamicStoreCopyConsoleUser(None, None, None) or [None])[0];
username = [username, ""][username in [u"loginwindow", None,
u""]]; sys.stdout.write(username + "\n");')
```

Check your sh scripts!

also check for Perl, Python, and Ruby one-liners

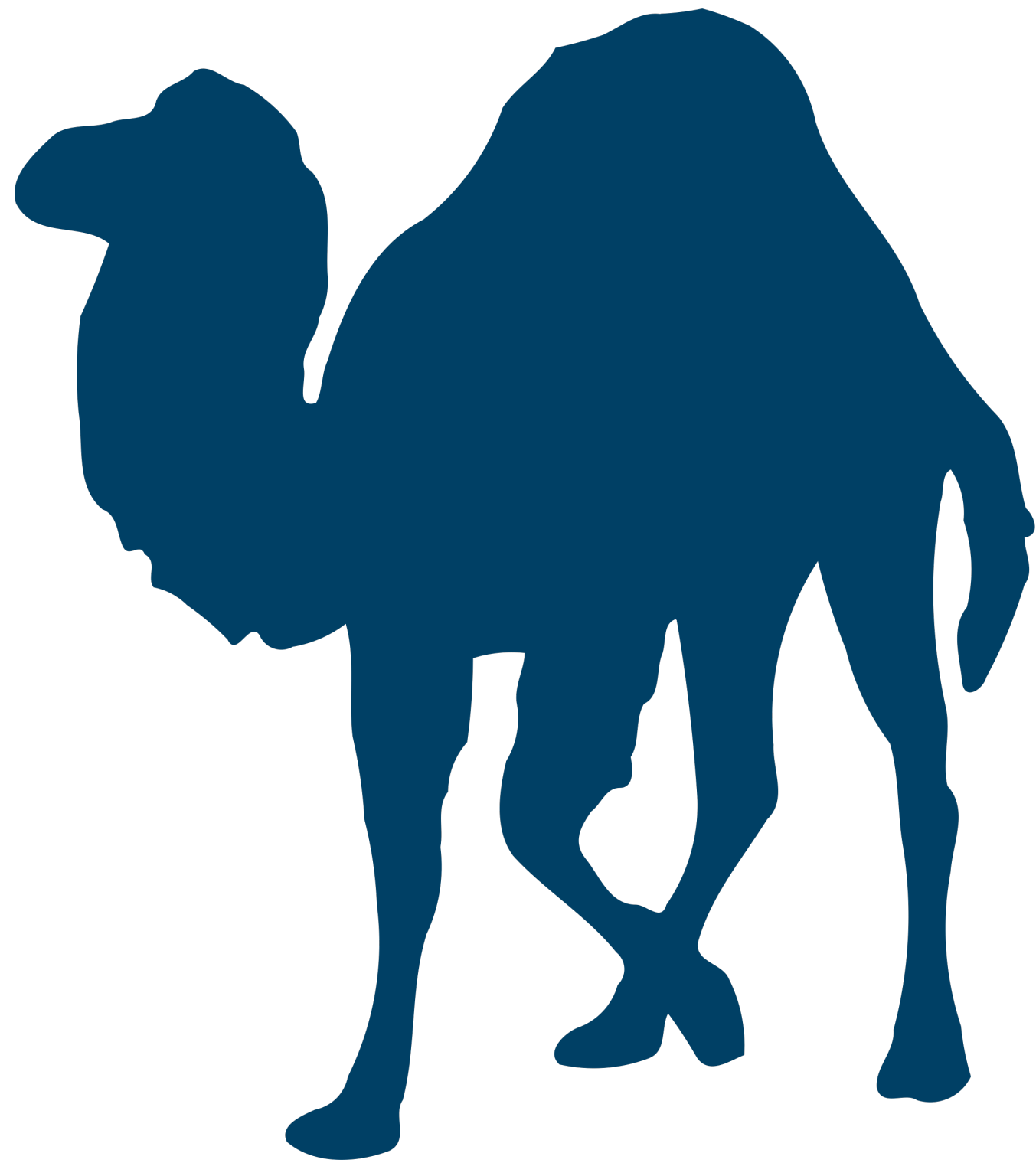
```
LoggedInUser=$( echo "show State:/Users/ConsoleUser" | scutil |  
awk '/Name :/ && ! /loginwindow/ { print $3 }' )
```



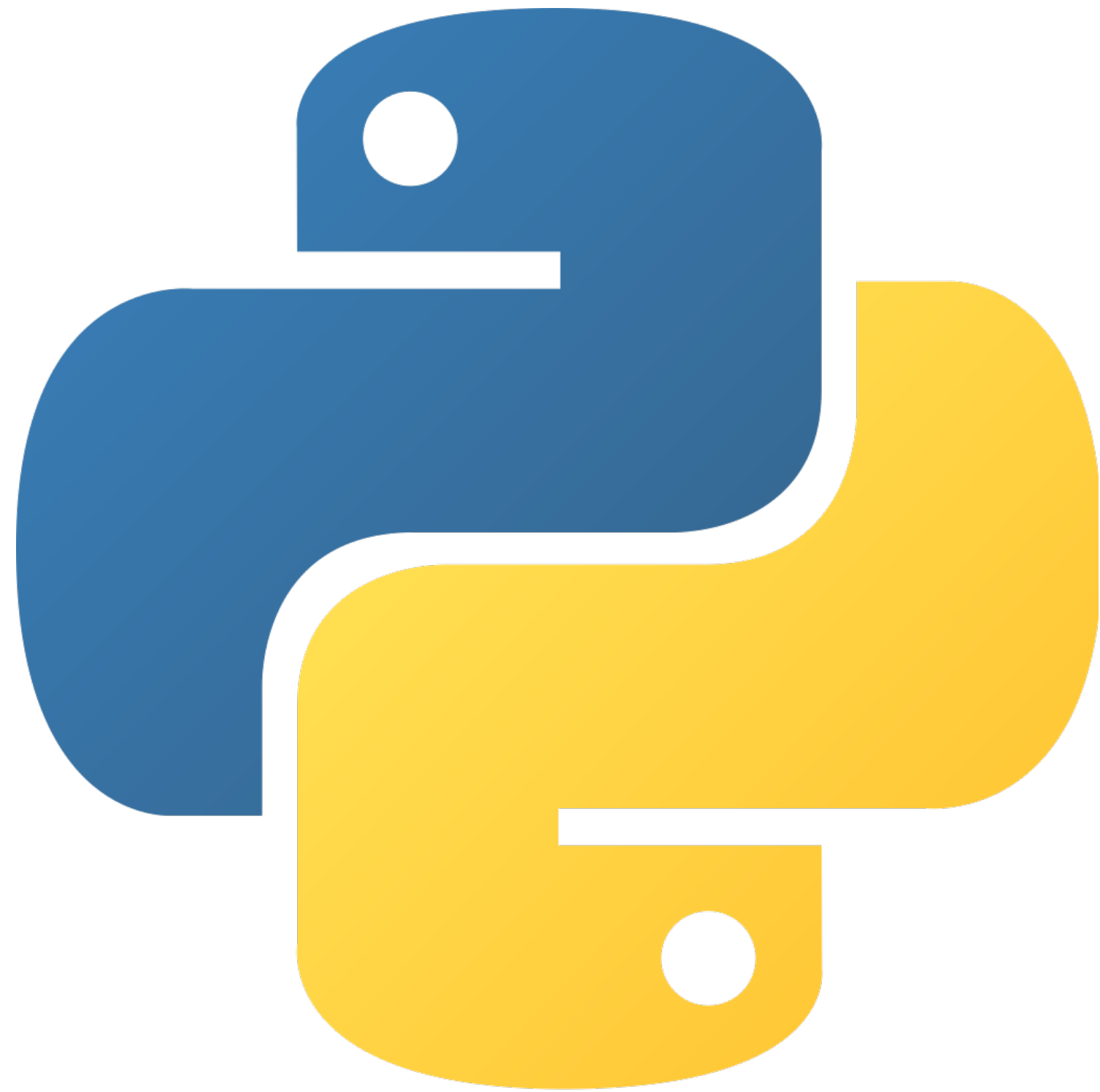

Scripting Language Runtimes

Deprecations

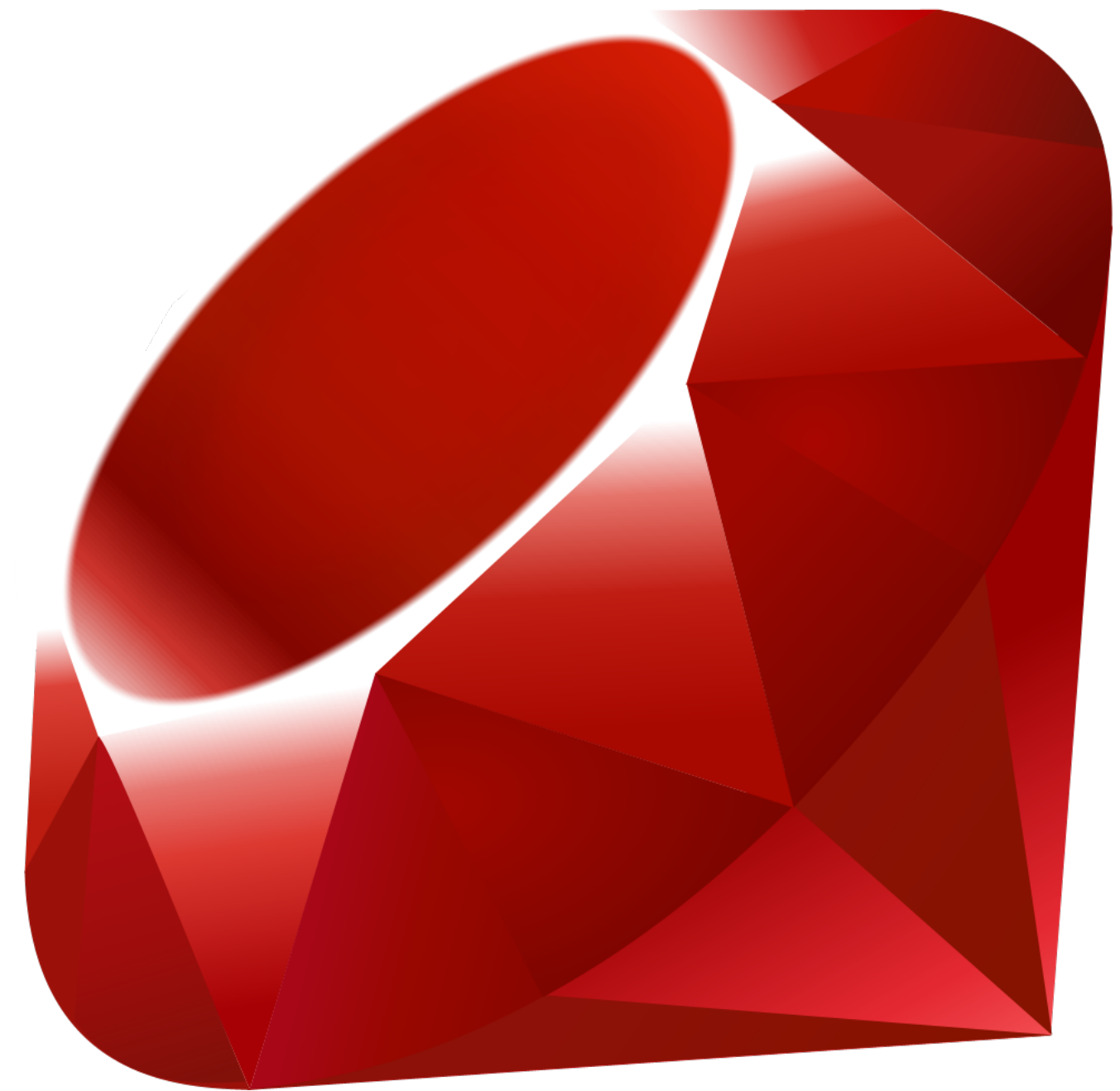
- Scripting language runtimes such as Python, Ruby, and Perl are included in macOS for compatibility with legacy software. Future versions of macOS won't include scripting language runtimes by default, and might require you to install additional packages. If your software depends on scripting languages, it's recommended that you bundle the runtime within the app. (49764202)
- Use of Python 2.7 isn't recommended as this version is included in macOS for compatibility with legacy software. Future versions of macOS won't include Python 2.7. Instead, it's recommended that you run `python3` from within Terminal. (51097165)



Perl



Python



Ruby

Don't Panic! (yet)

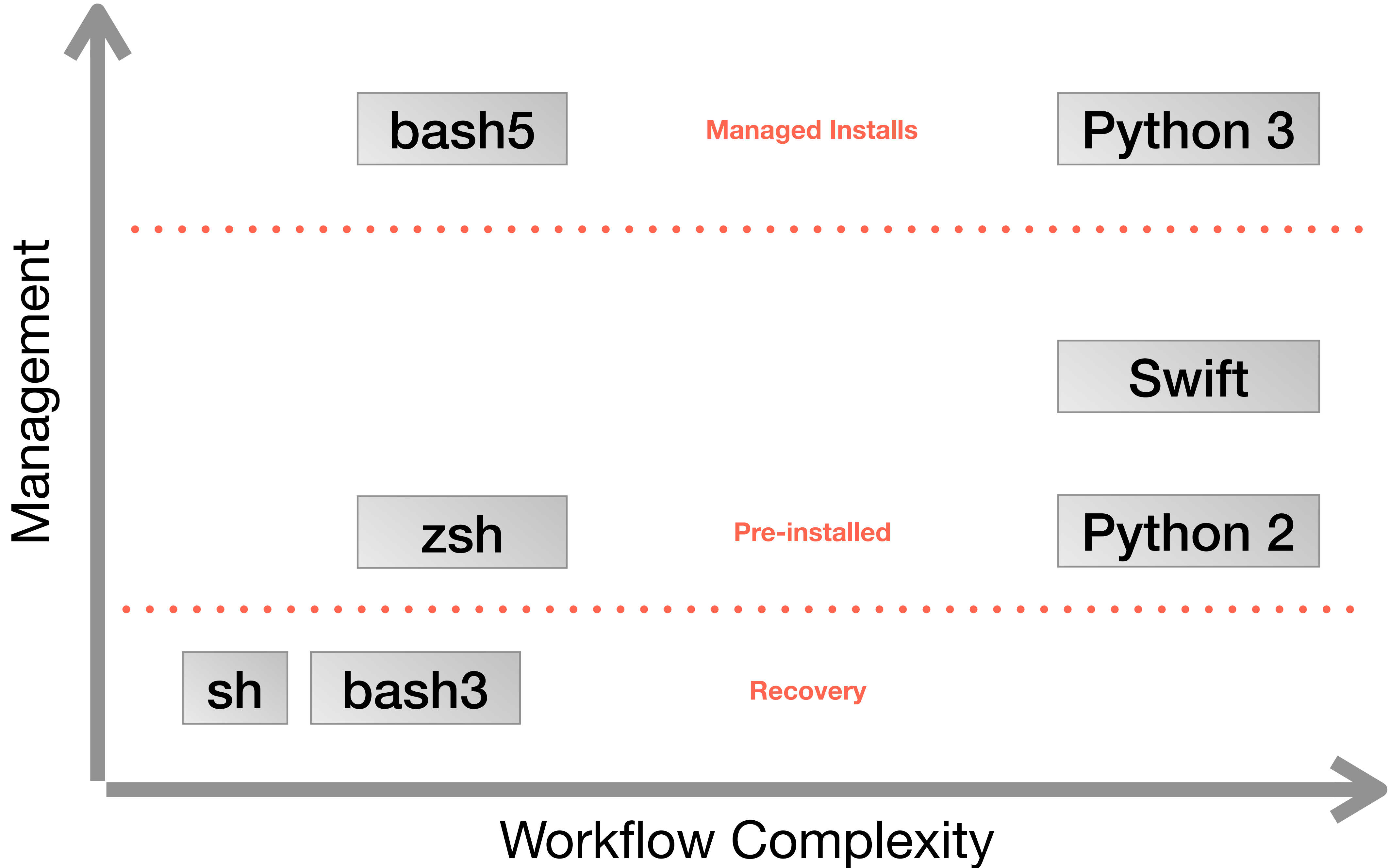
Perl, Python, and Ruby still in Catalina 🌴

Just like `/bin/bash v3`

Python 2.7 is deprecated, no more updates starting 2020

“Future macOS”

Scripting "Future macOS"



Scripting "Future macOS"

automation and workflows

zsh

installation scripts

sh

complex projects and data

Swift (compiled)

install and manage

Python 3, Ruby, Perl, etc.

Check your Package Scripts

```
% ./pkgcheck.sh SamplePkgs/SourceCodePro-2.030d.pkg
```

```
SourceCodePro-2.030d
```

```
SamplePkgs/SourceCodePro-2.030d.pkg
```

```
Signature:          None
```

```
Notarized:         No, no usable signature
```

```
Type:             Flat Component PKG
```

```
Identifier:       com.example.SourceCodePro
```

```
Version:         2.030d
```

```
Location:        /
```

```
Contains 2 resource files
```

```
postinstall has shebang #!/bin/bash
```

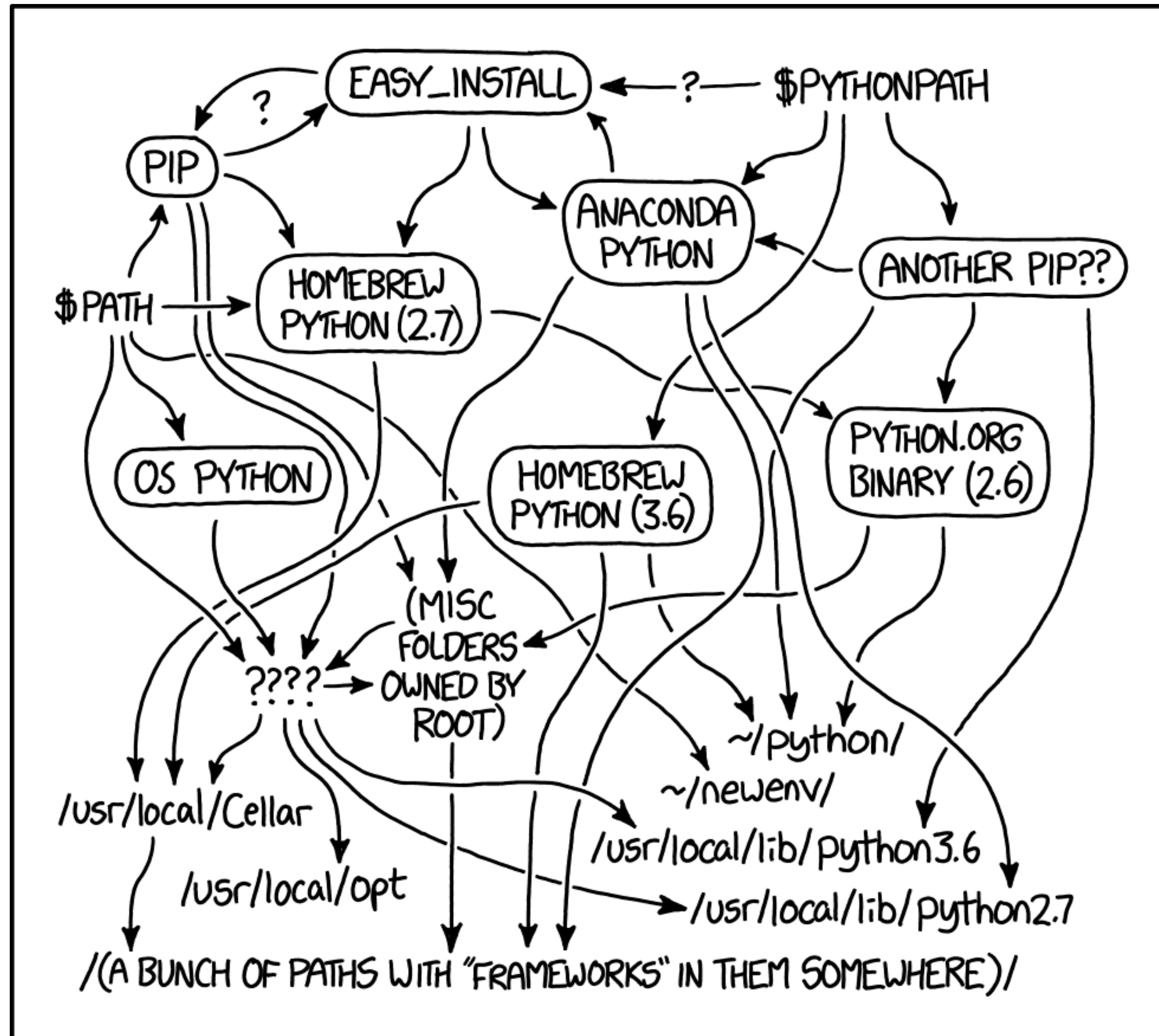
```
preinstall has shebang #!/bin/bash
```

Manage the run-time

Install vendor packages

Build your own installer packages

Install to custom location
to avoid conflicts with user versions



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Script files



hello.py



hello.sh

Swift – Script files

```
#!/usr/bin/swift
```

```
import Foundation
```

```
// first argument is path to binary, drop it
```

```
let arguments = CommandLine.arguments.dropFirst()
```

```
if arguments.count > 0 {
```

```
    let argumentString = arguments.joined(separator: " ")
```

```
    print("Hello, \(argumentString)")
```

```
} else {
```

```
    print("Hello, anybody!")
```

```
}
```

```
exit(0)
```

Swift – Script files



The “swift” command requires the command line developer tools. Would you like to install the tools now?

Choose Install to continue. Choose Get Xcode to install Xcode and the command line developer tools from the App Store.

Get Xcode

Not Now

Install

% chr

% ./l

Hello

Swift – Script files

```
#!/usr/bin/swift

import Foundation

// first argument is path to binary, drop it
let arguments = CommandLine.arguments.dropFirst()

if arguments.count > 0 {
    let argumentString = arguments.joined(separator: " ")
    print("Hello, \(argumentString)")
} else {
    print("Hello, anybody!")
}

exit(0)
```

Swift 5

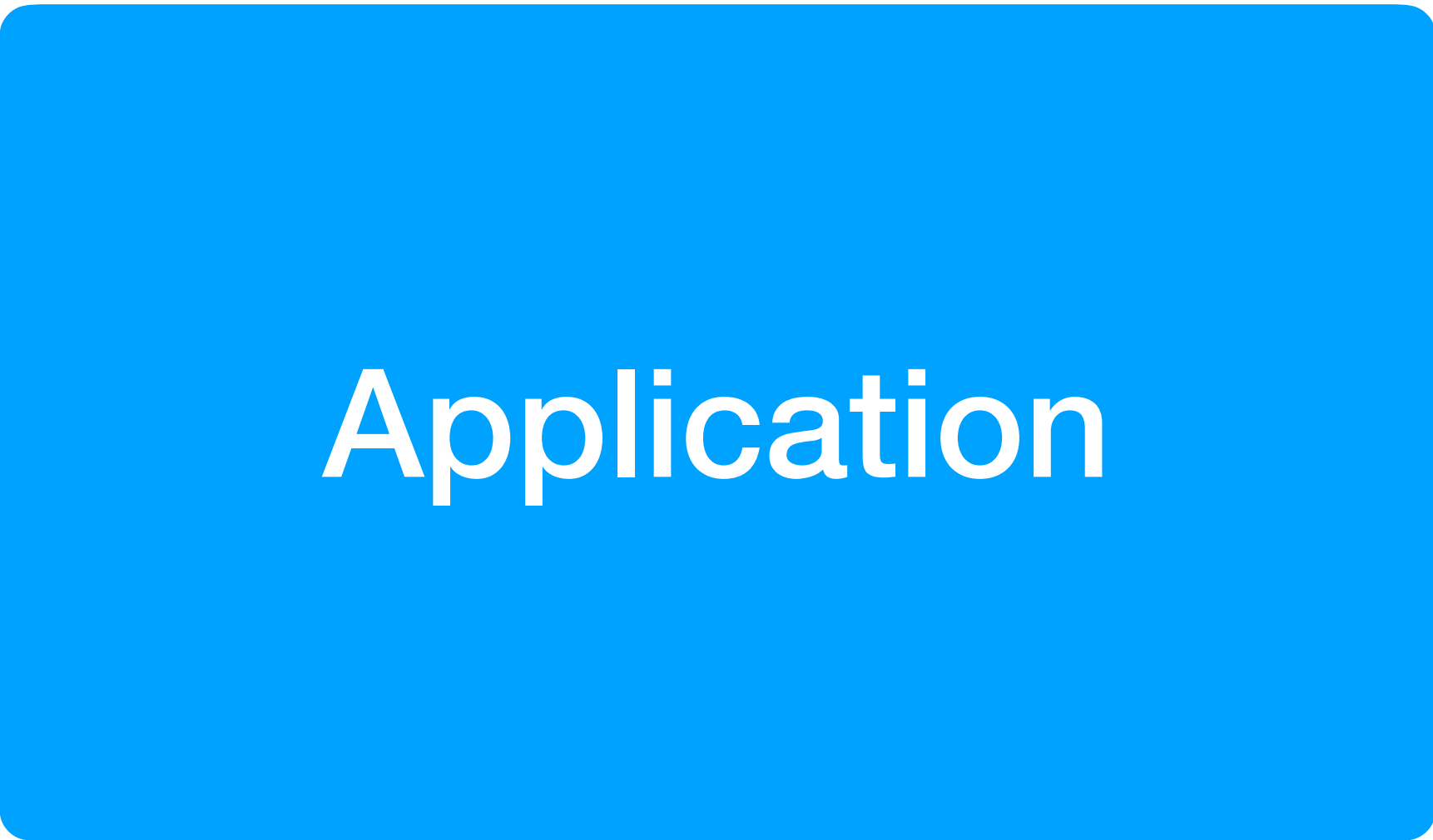
Swift 5 – ABI stability

ABI: Application Binary Interface

Use libraries built by a different compiler version

Compiler Version 4.0

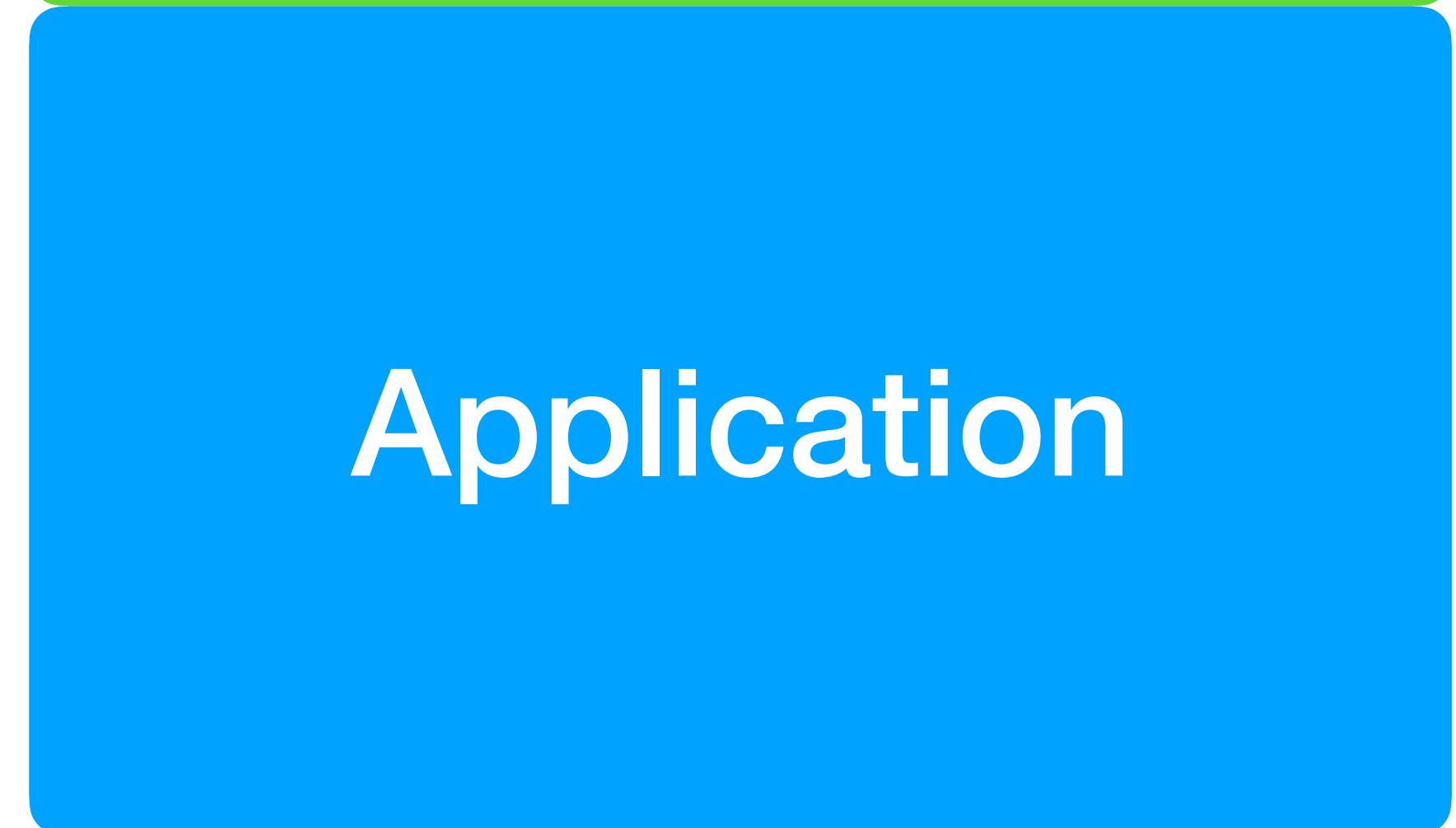
Compiler Version 4.2



Combined Binary

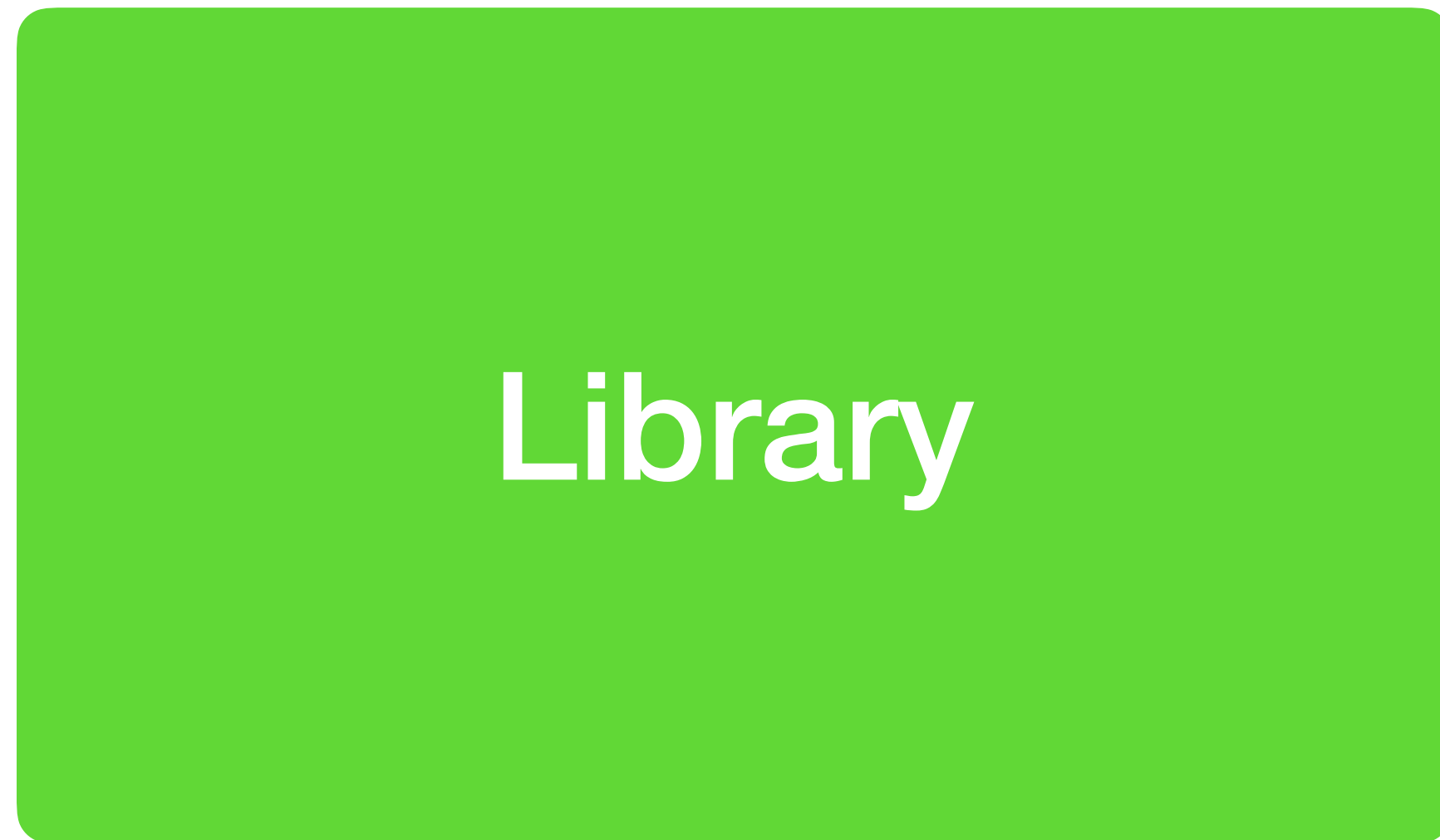


hello
Unix executable - 10,2 MB



Version 5

Compiler Version 5.1



Included in macOS 10.14.4+

hello
Unix executable - 37 KB





Download Swift 5 Runtime Support for Command Line Tools

[Download](#)

Starting with Xcode 10.2, Swift 5 command line programs you build require the Swift 5 runtime support libraries built into macOS. These libraries are included in the OS starting with macOS Mojave 10.14.4. When running on earlier versions of macOS, this package must be installed to provide the necessary Swift 5 libraries. This package is not necessary for apps with graphical user interfaces.

Post Date: Mar 25, 2019

File Size: 3.2 MB

Swift 5+ Command Line Tools

10.14.4+: 👍 runtime support included in macOS

<10.14.4: 📦 requires Swift Runtime Support

Required for Notarization

Managing Swift Tools

Install Swift Runtime Support <10.14.4

Compile and Install binary tools

Sign and Notarize tools

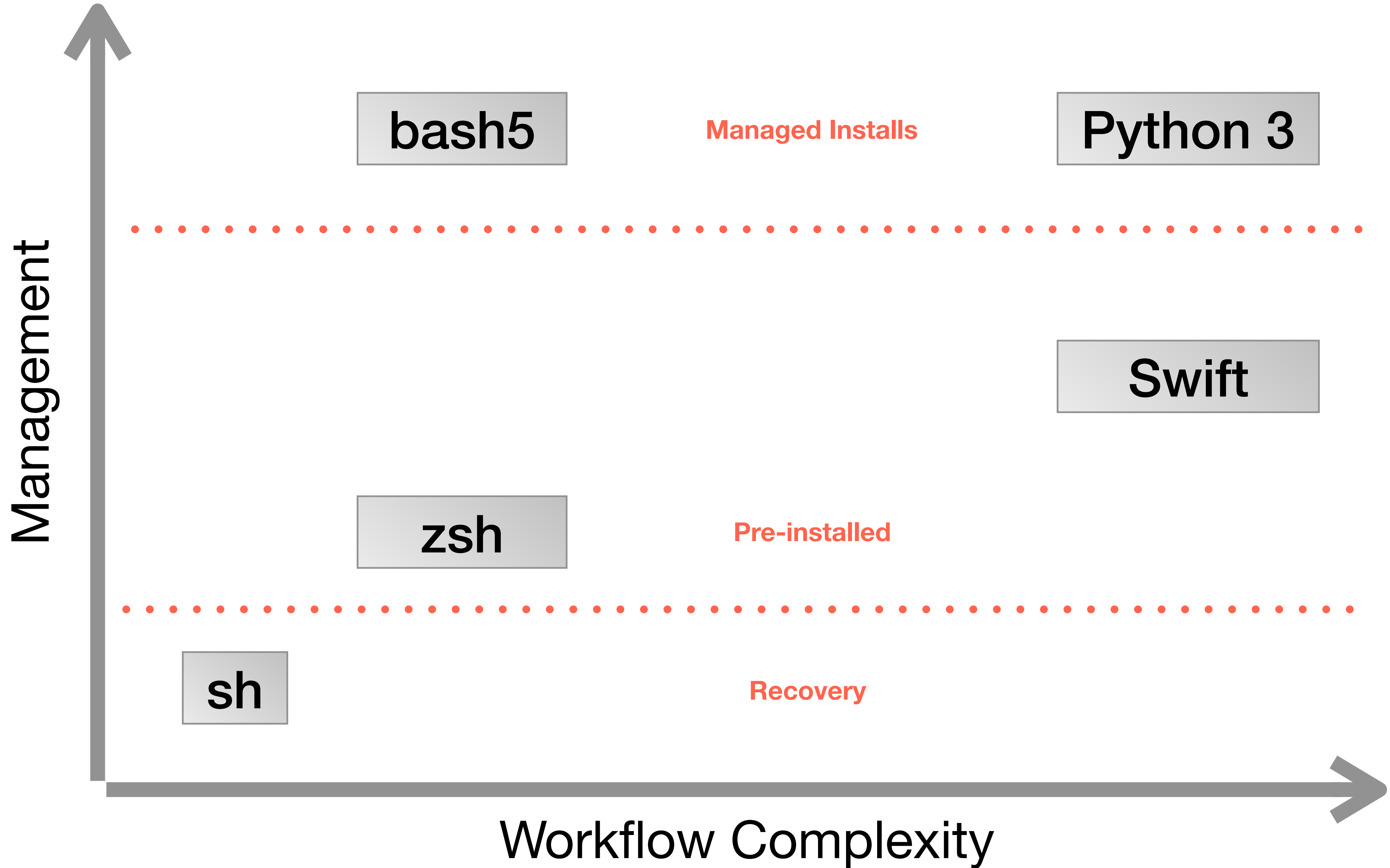
Signing and Notarization

Signing required for PPC whitelisting

Gatekeeper checks Notarization

Automated Installations bypass Gatekeeper

"Future macOS"?



Conclusion

zsh

"zee-shell" 

bash v3 is deprecated

zsh is not that different

Check sh scripts for bashisms

Check scripts for Perl, Python, and Ruby one-liners

Scripting "Future macOS"

Scripting "Future macOS"

automation and workflows

zsh

installation scripts

sh

complex projects and data

Swift (compiled)

install and manage

Python 3, Ruby, Perl, etc.

Don't Panic!

But start preparing...

scriptingosix.com/zsh



Thank you, Tycho



scriptingosx.com/zsh

