



# Cheat sheet

```
git checkout -b myBranch
```

(make changes)

```
git checkout master
```

```
git pull
```

```
git merge myBranch
```

Fix conflicts

```
git push
```

# Overview

- Collaborative development
- Version Control
- Git
- Methods
- Best practices

# Objectives

- Know about different version control systems.
- Understand why git is the best.
- Don't be a jerk to your group members.

# Collaborative development

- Software development model
- Focus on availability and communication
- Started with the linux kernel in '95
- Enables mass peer review
- Facilitates specialization
- Involves users

# Collaborative development

- Not unique to software development.
- **Your chosen profession.**
- How all good things are built.

**IF YOU WANT TO GO FAST, GO  
ALONE**



**IF YOU WANT TO GO FAR, GO  
TOGETHER.**





# linux runs everything

- your tv
- your car
- your bank
- your phone
- air traffic control
- nuclear submarines
- most of the global economy

# seriously...

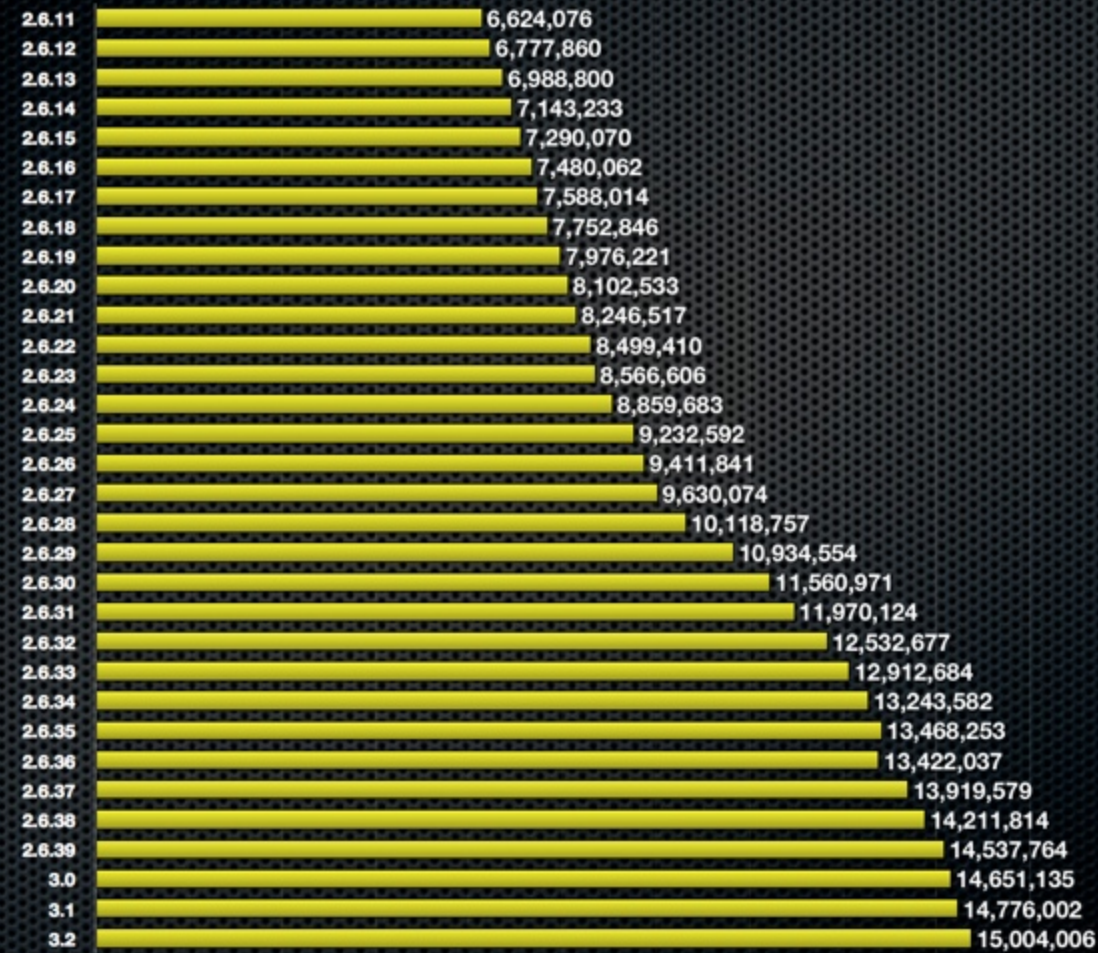
- 1.3 million smart phones
- 700,000 televisions
- 92% of the worlds high performance computing systems: weather, cern, nasa
- 85% of the global economic trading systems

■ ■ ■

- facebook
- google
- amazon
- twitter
- apple

## Number of lines of code in the Linux kernel

Linux kernel version



Data source: Linux Foundation

[www.pingdom.com](http://www.pingdom.com)

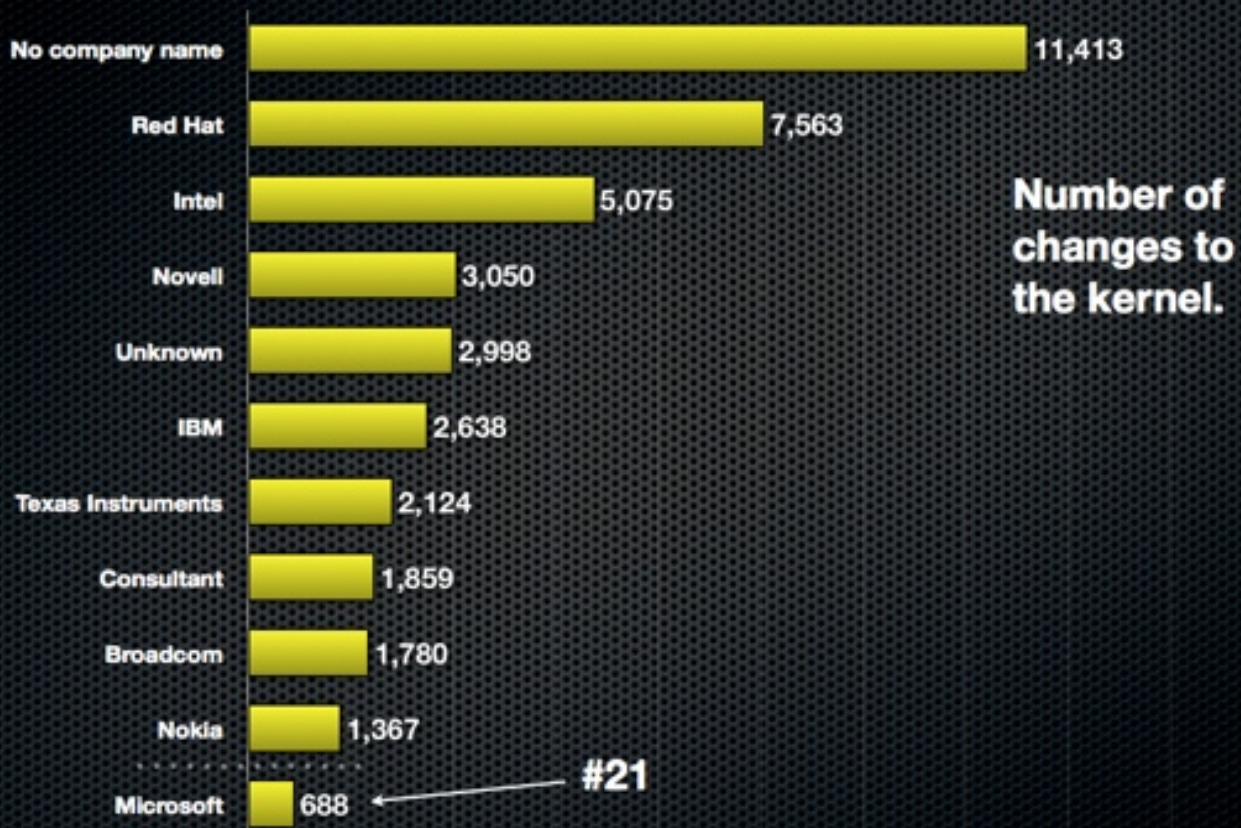
Linux was built gradually  
and collaboratively.

kernel version 3.2.

1,316 developers

226 known companies

## Top 10 contributors to the Linux kernel since version 2.6.36



Data source: Linux Foundation

[www.pingdom.com](http://www.pingdom.com)

This wouldn't work  
without a system.



# Linus Torvalds

- Created the linux kernel.
- Linux kernel maintenance changes to the software were passed around as patches and archived files.
- Used bit keeper (DVCS) for a while.
- Made git
- Today about 6% of the linux kernel is Linus's code.

Many eyeballs  
make all bugs  
shallow



VCS

# synonyms

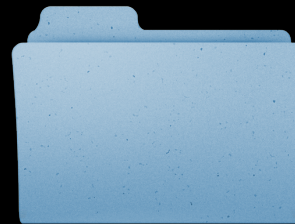
- source control
- revision control
- version control
- source code control systems
- distributed version control systems

# network types

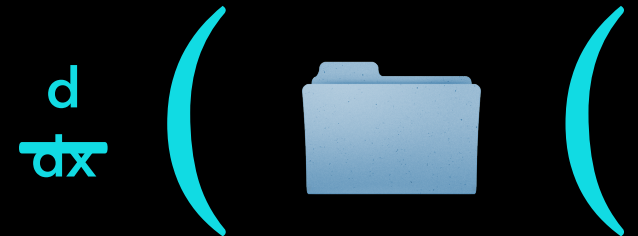
- none



- centralized



- distributed



# no network

- one file at a time
- locks a file when accessed
- one central development environment
- one shared point of failure

▸ RCS, SCCS



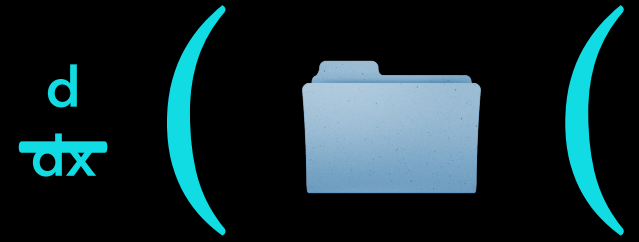
# centralized



- multiple files at a time
- stores change sets irrespective of their files
- one central development environment
- one shared point of failure

▸ SourceSafe, Subversion, CVS

# distributed

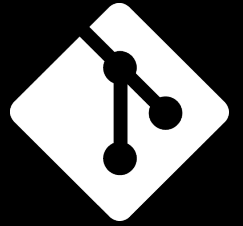


- everyone has the whole repository
- change sets
- independent, local development environments
- available offline

▸ Bitkeeper, Bazaar, Git, Mercurial

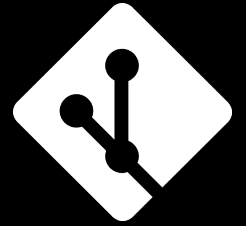


# branches



- everyone has the whole repository
- safe, independent development environments

# merging branches



- recursive
  - normal merge
- octopus
  - merges multiple heads in one commit as long as it can do it cleanly.
- ours
  - keeps the history of a branch without any of the effects of the branch.
- subtree
  - merge in another project into a subdirectory of your current project.

# git vs. subversion

- Git is much faster
- Subversion allows you to check out just a subtree of a repository; Git requires you to clone the entire repository.
- Git's repositories are ~30x smaller.
- Git branches are simpler and require fewer resources.
- Git branches carry their entire history.
- Subversion's UI is more mature than Git's.
- Git provides better auditing of branch and merge events.

# git is awesome

- distributed
- flexible
- secure
- simple
- local
- light-weight
- fast

# distributed

- each individual machine is a development environment and has the whole repository
- allows for offline development

# flexible

- Many different work flows.
- Adapts to your needs.
- Various intelligent merge methods.

# secure

- Stores change sets as hashes.
- requires authentication for pushing and pulling.

# simple

- You guys did Homework 1.
- That's most of it.



# local

- Everyone has their own independent development environment.
- Break the build all day long, your team mates won't notice.

# light-weight

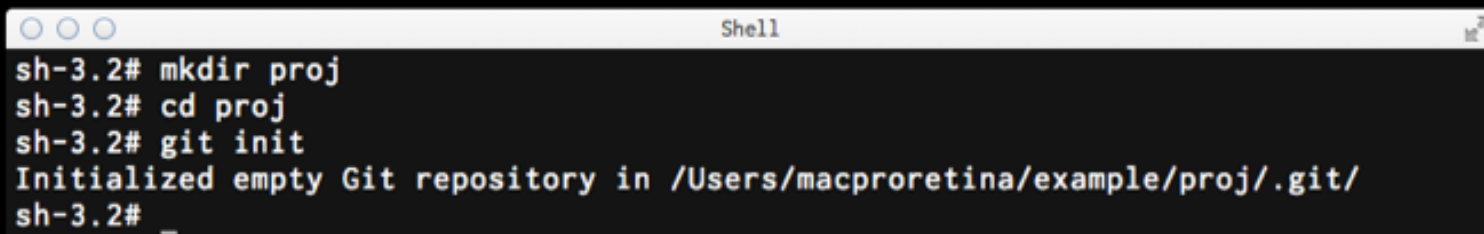
- Mozilla's git repository is ~30x smaller than the same content in SVN.

# fast

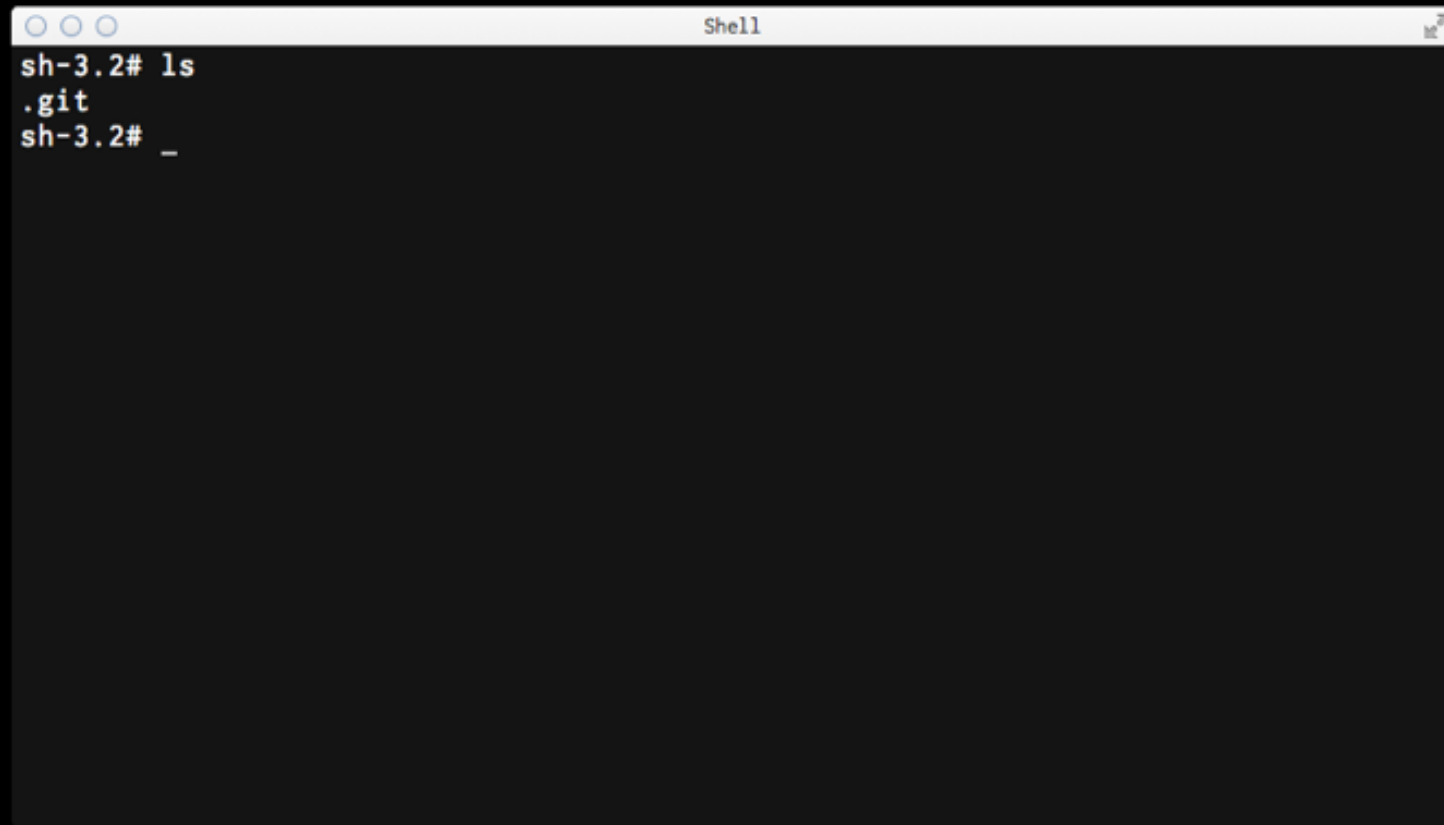
- Nearly every action in git is considerably faster than it's analogue in other version control systems.
- Change sets are minimized and hashed and packaged before pushing / pulling

methods

# git init

A terminal window titled "Shell" with three window control buttons (red, yellow, green) in the top-left corner. The terminal shows a sequence of commands and their output: "sh-3.2# mkdir proj", "sh-3.2# cd proj", "sh-3.2# git init", and the output "Initialized empty Git repository in /Users/macproretina/example/proj/.git/". The prompt "sh-3.2#" is followed by an underscore "\_" on the last line.

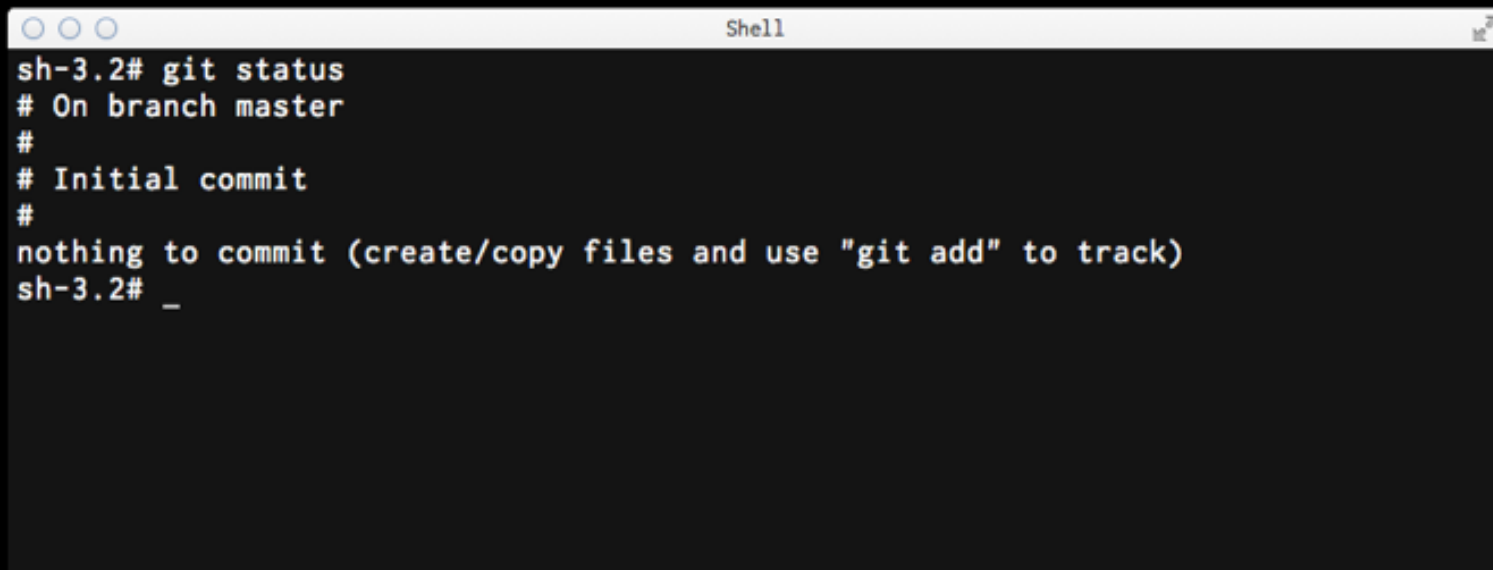
```
sh-3.2# mkdir proj
sh-3.2# cd proj
sh-3.2# git init
Initialized empty Git repository in /Users/macproretina/example/proj/.git/
sh-3.2# _
```



```
sh-3.2# ls
.git
sh-3.2# _
```

A terminal window titled "Shell" with three window control buttons (minimize, maximize, close) on the left. The terminal displays the command "ls" and its output ".git". The prompt "sh-3.2#" is shown before and after the command. A cursor "\_" is visible on the line following the command.

# git status

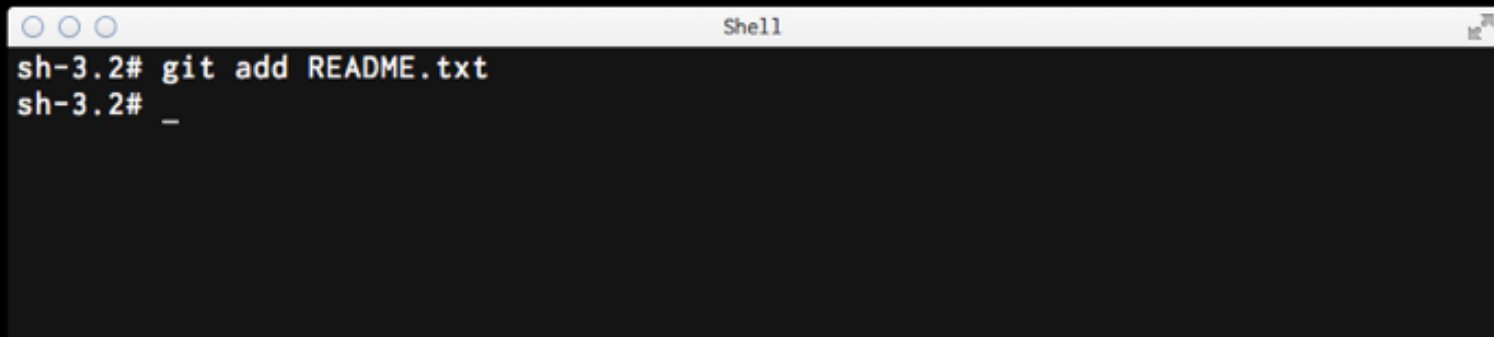
A terminal window titled "Shell" with three window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the "git status" command. The text is as follows:

```
sh-3.2# git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
sh-3.2# _
```

```
Shell
sh-3.2# ls
.git                README.txt
sh-3.2# git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README.txt
nothing added to commit but untracked files present (use "git add" to track)
sh-3.2# _
```



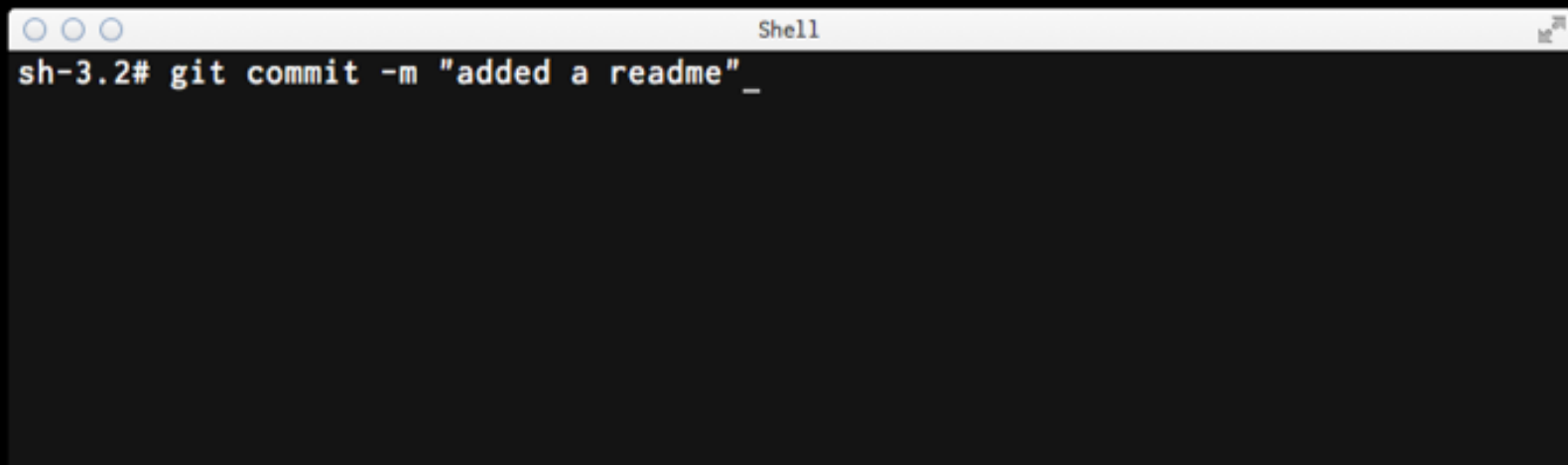
# git add

A terminal window titled "Shell" with three window control buttons (red, yellow, green) in the top-left corner. The terminal shows the command "git add README.txt" being executed in a shell prompt "sh-3.2#". The prompt "sh-3.2#" is followed by a cursor line consisting of an underscore and a space character.

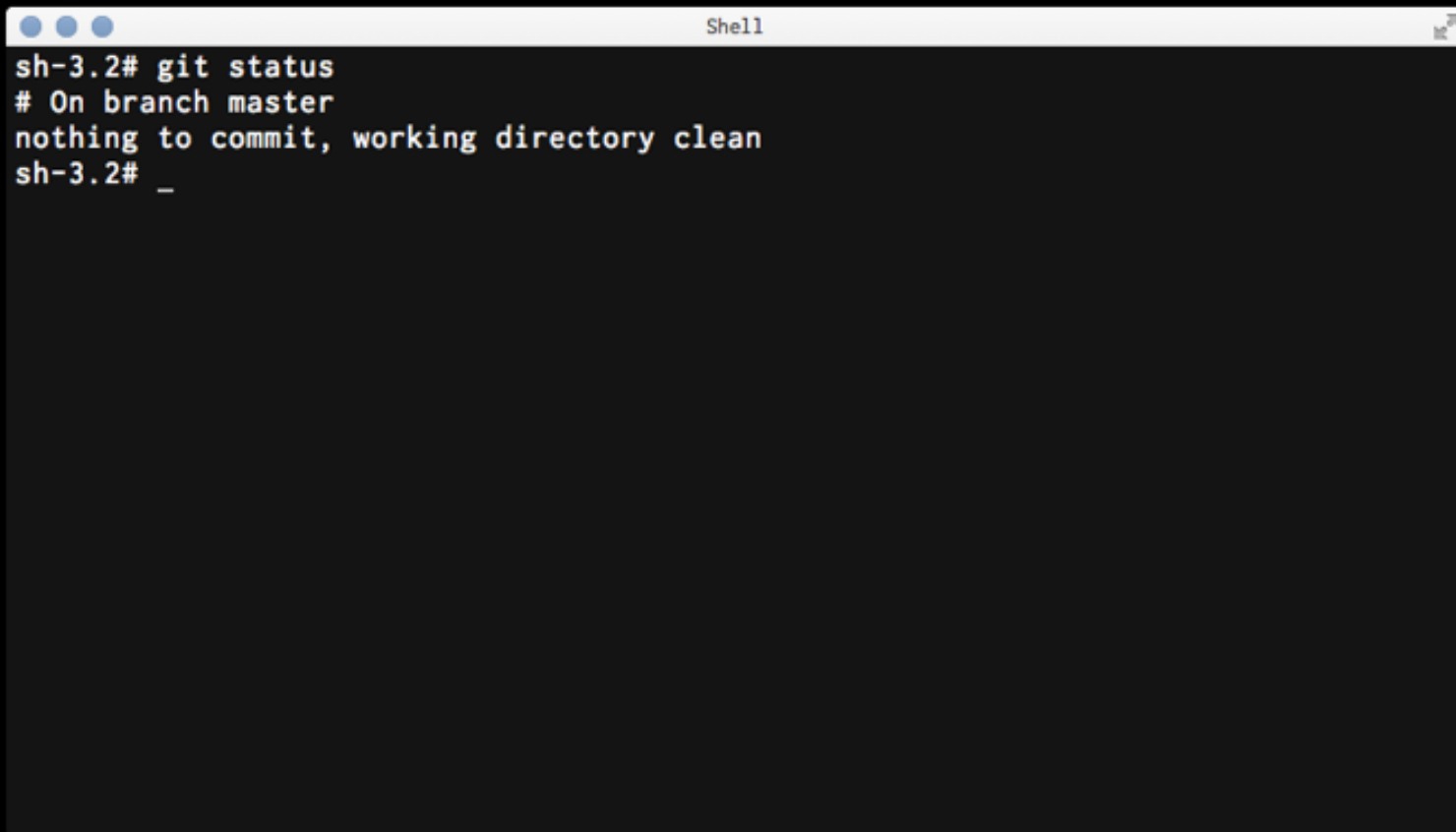
```
sh-3.2# git add README.txt
sh-3.2# _
```

```
Shell
sh-3.2# git add README.txt
sh-3.2# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README.txt
#
sh-3.2# _
```

# git commit

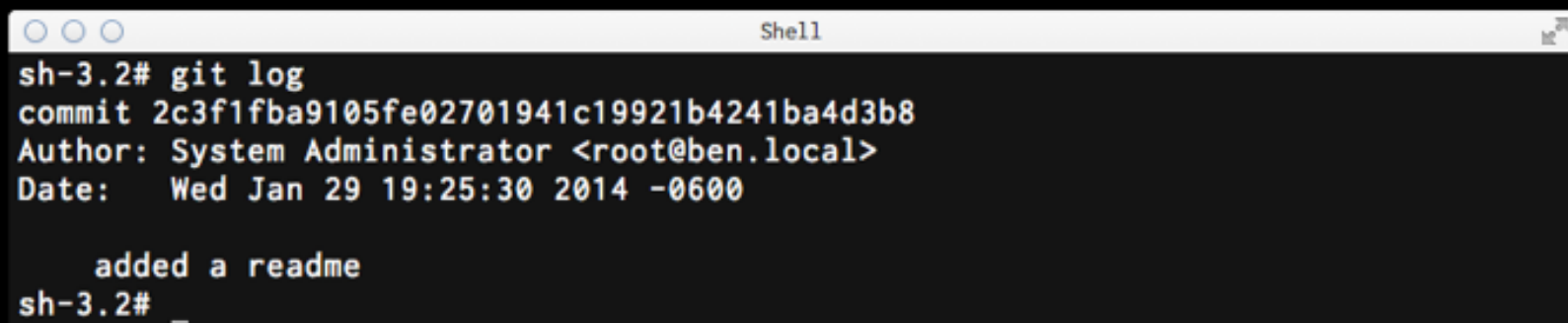
A terminal window with a title bar containing three window control buttons and the text "Shell". The terminal content shows a shell prompt "sh-3.2#" followed by the command "git commit -m 'added a readme'" and a cursor. The terminal background is dark gray.

```
sh-3.2# git commit -m "added a readme" _
```



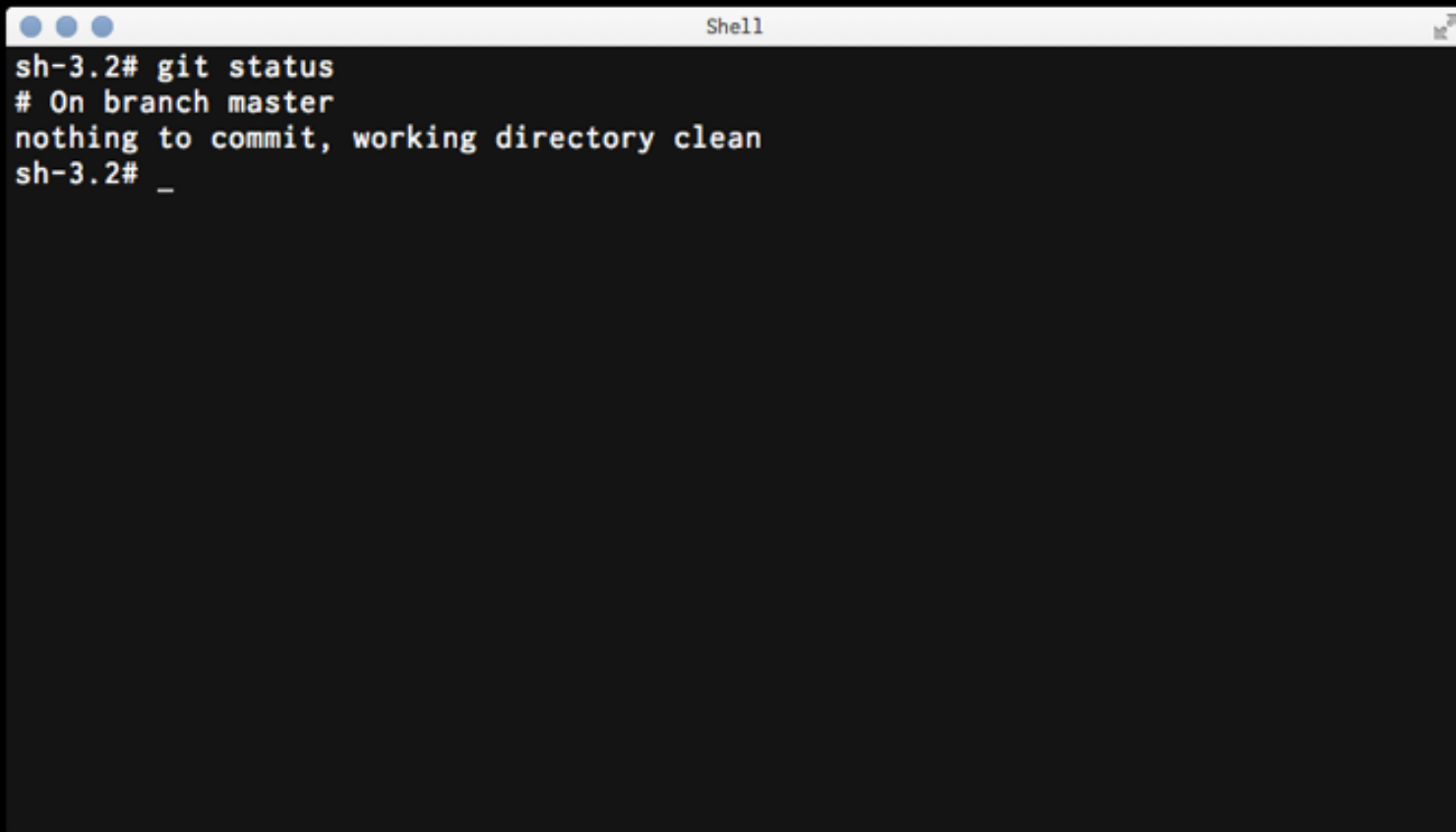
```
sh-3.2# git status
# On branch master
nothing to commit, working directory clean
sh-3.2# _
```

# git log

A terminal window titled "Shell" with three window control buttons (minimize, maximize, close) in the top-left corner. The terminal shows the output of the "git log" command. The output includes the commit hash "2c3f1fba9105fe02701941c19921b4241ba4d3b8", the author "System Administrator <root@ben.local>", the date "Wed Jan 29 19:25:30 2014 -0600", and the commit message "added a readme". The prompt "sh-3.2#" is visible at the bottom, followed by an underscore character "\_".

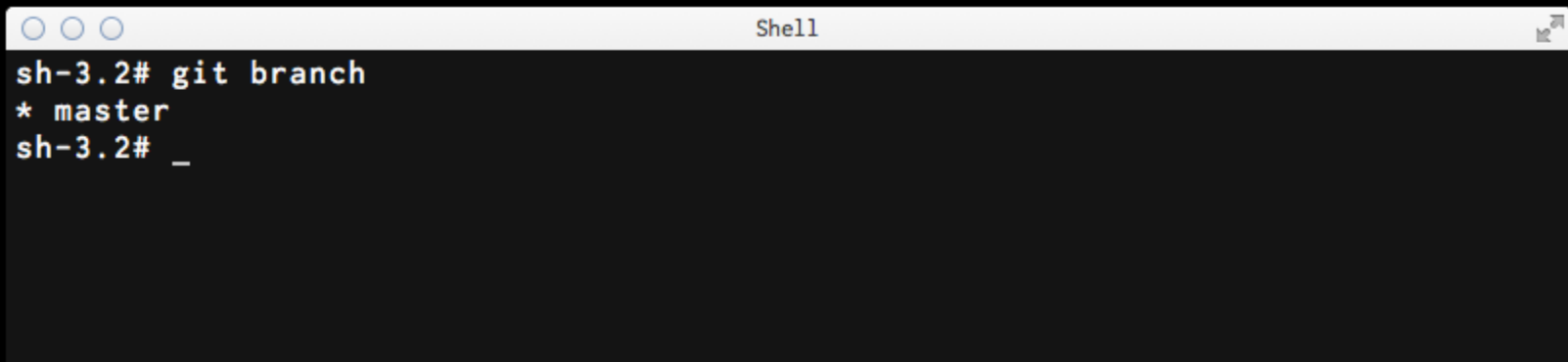
```
sh-3.2# git log
commit 2c3f1fba9105fe02701941c19921b4241ba4d3b8
Author: System Administrator <root@ben.local>
Date:   Wed Jan 29 19:25:30 2014 -0600

    added a readme
sh-3.2# _
```



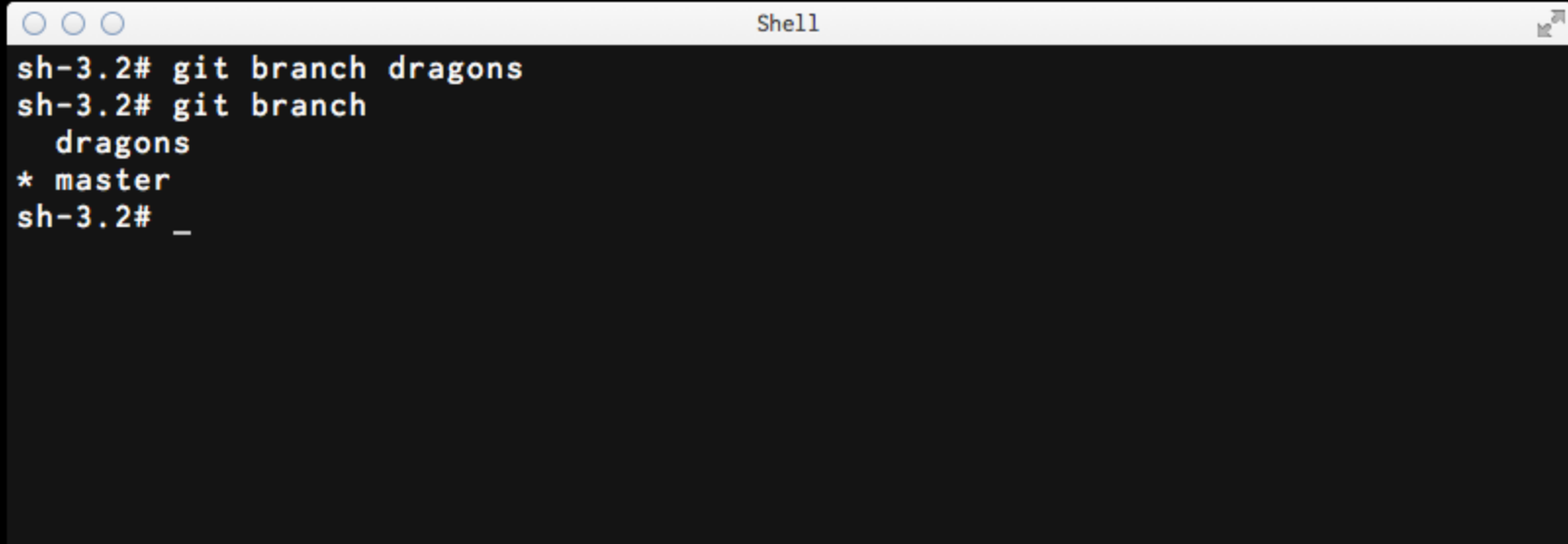
```
sh-3.2# git status
# On branch master
nothing to commit, working directory clean
sh-3.2# _
```

# git branch

A terminal window titled "Shell" with three window control buttons (red, yellow, green) in the top-left corner. The terminal shows the command "git branch" being executed, which returns "\* master". The prompt "sh-3.2#" is visible before and after the output.

```
sh-3.2# git branch
* master
sh-3.2# _
```

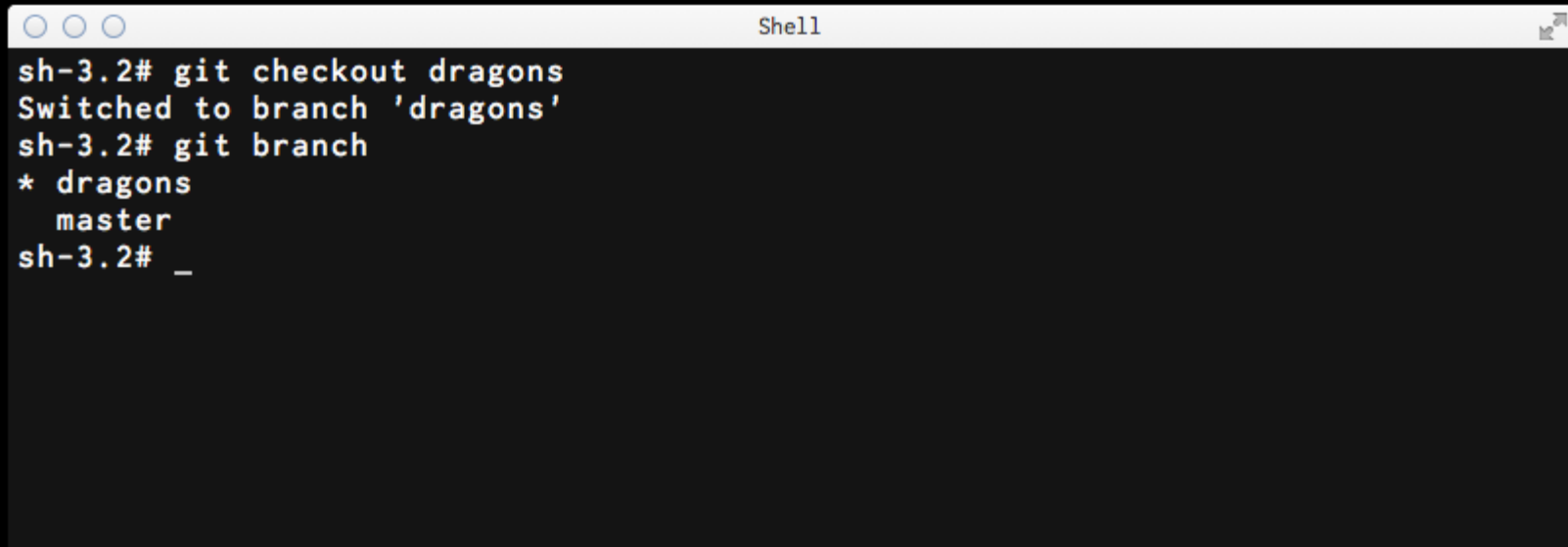
# git branch dragons

A terminal window titled "Shell" with three window control buttons (minimize, maximize, close) in the top-left corner. The terminal shows the execution of two git commands: "git branch dragons" and "git branch", followed by the output of the second command.

```
sh-3.2# git branch dragons
sh-3.2# git branch
  dragons
* master
sh-3.2# _
```

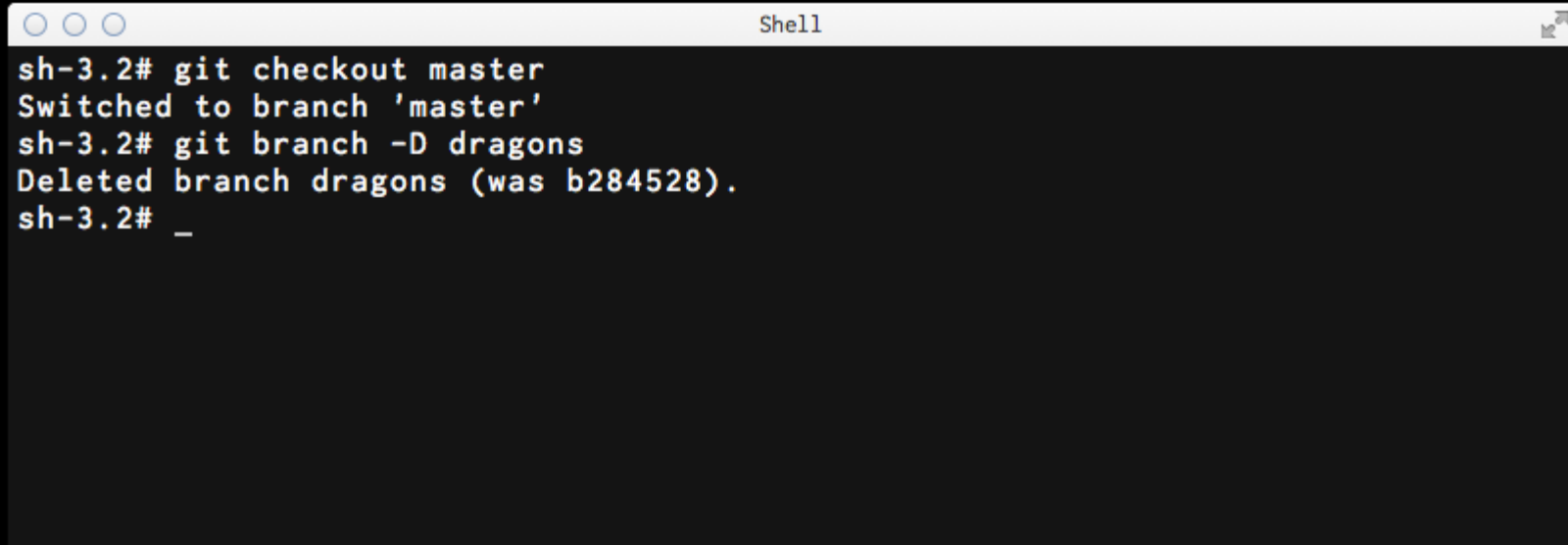


# git checkout dragons

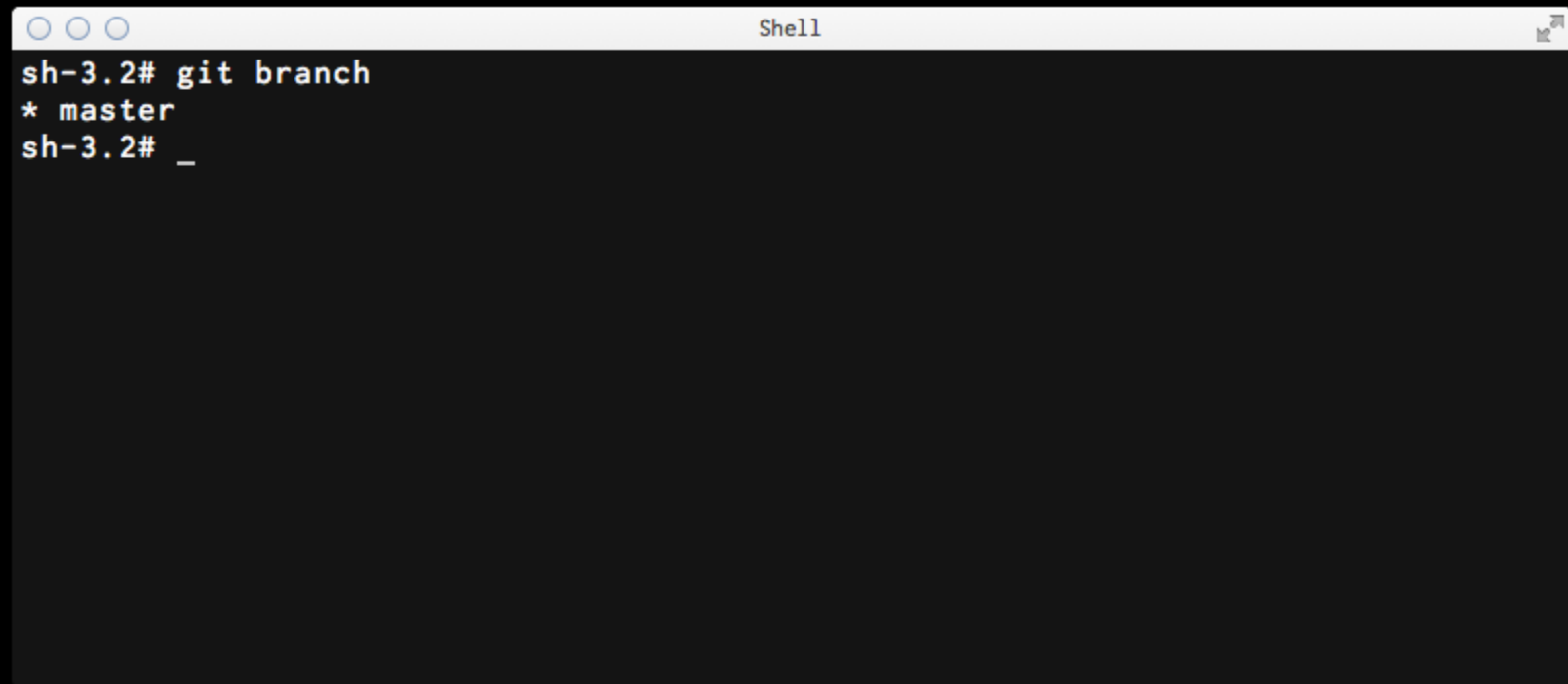


```
Shell
sh-3.2# git checkout dragons
Switched to branch 'dragons'
sh-3.2# git branch
* dragons
  master
sh-3.2# _
```

# git branch -D dragons

A terminal window titled "Shell" with three window control buttons (red, yellow, green) in the top-left corner. The terminal displays the following text:

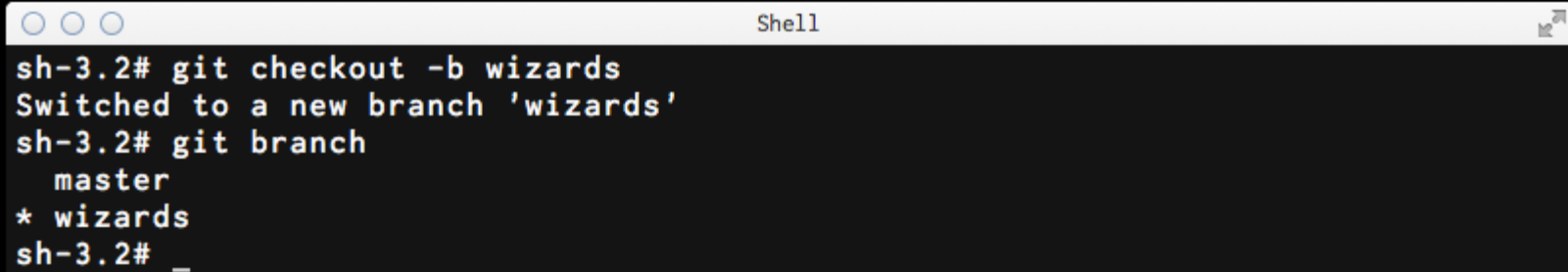
```
sh-3.2# git checkout master
Switched to branch 'master'
sh-3.2# git branch -D dragons
Deleted branch dragons (was b284528).
sh-3.2# _
```



```
sh-3.2# git branch
* master
sh-3.2# _
```

A terminal window titled "Shell" with three window control buttons (minimize, maximize, close) on the left and a close button on the right. The terminal content shows the command "git branch" being executed, resulting in the output "\* master". The prompt "sh-3.2#" is followed by an underscore "\_" on the next line.

# git checkout -b wizards



```
Shell
sh-3.2# git checkout -b wizards
Switched to a new branch 'wizards'
sh-3.2# git branch
  master
* wizards
sh-3.2# _
```

A terminal window titled "Shell" with three window control buttons (minimize, maximize, close) in the top-left corner. The terminal displays the execution of the `git checkout -b wizards` command, which successfully creates and switches to a new branch named 'wizards'. Subsequently, the `git branch` command is run, showing the current branch list with 'master' and 'wizards' (the active branch, indicated by an asterisk). The prompt returns to `sh-3.2#` followed by an underscore.

best practices

```
git checkout -b myBranch
```

(make your changes)

```
git checkout master
```

```
git pull origin master
```

```
git merge myBranch
```

(fix conflicts if they exist)

```
git push origin master
```

- Split up your development to avoid merge conflicts.
- Don't commit a broken build.
- Work in branches.
- Comment your commits.



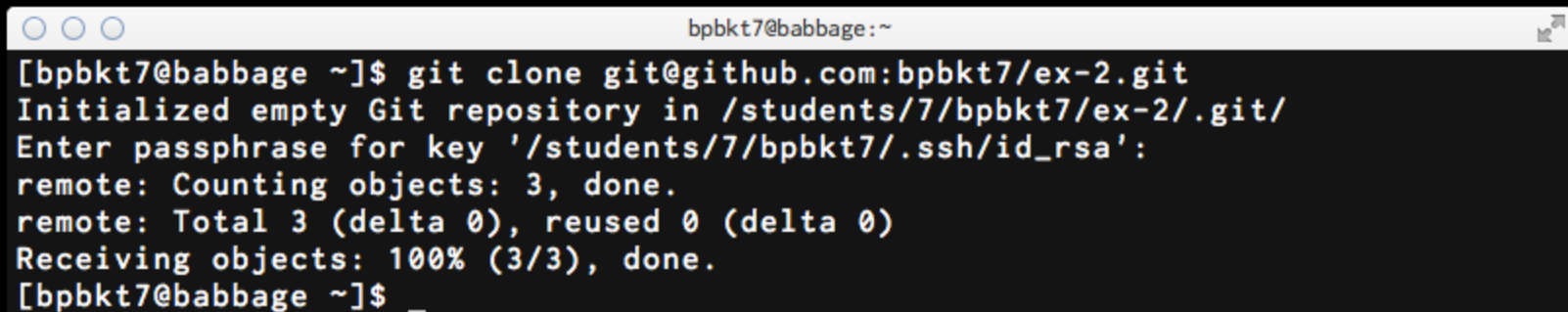
bpbkt7@babbage:~/test



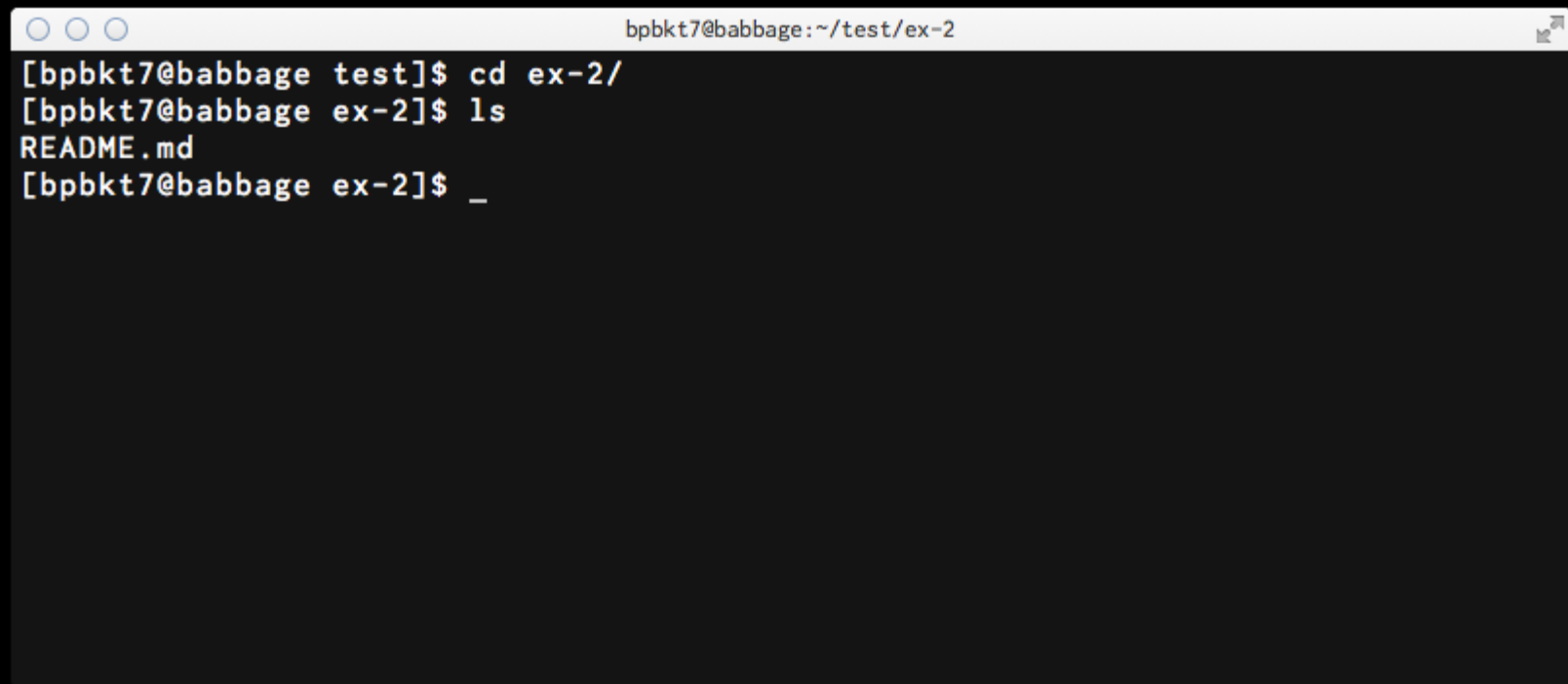
```
[bpbkt7@babbage test]$ ls  
[bpbkt7@babbage test]$ _
```



# git clone

A terminal window with a title bar showing three window control buttons on the left and the text 'bpbkt7@babbage:~' on the right. The terminal content shows the execution of the 'git clone' command to clone a repository from GitHub. The output includes messages about initializing the repository, entering a passphrase for an SSH key, and the progress of downloading objects.

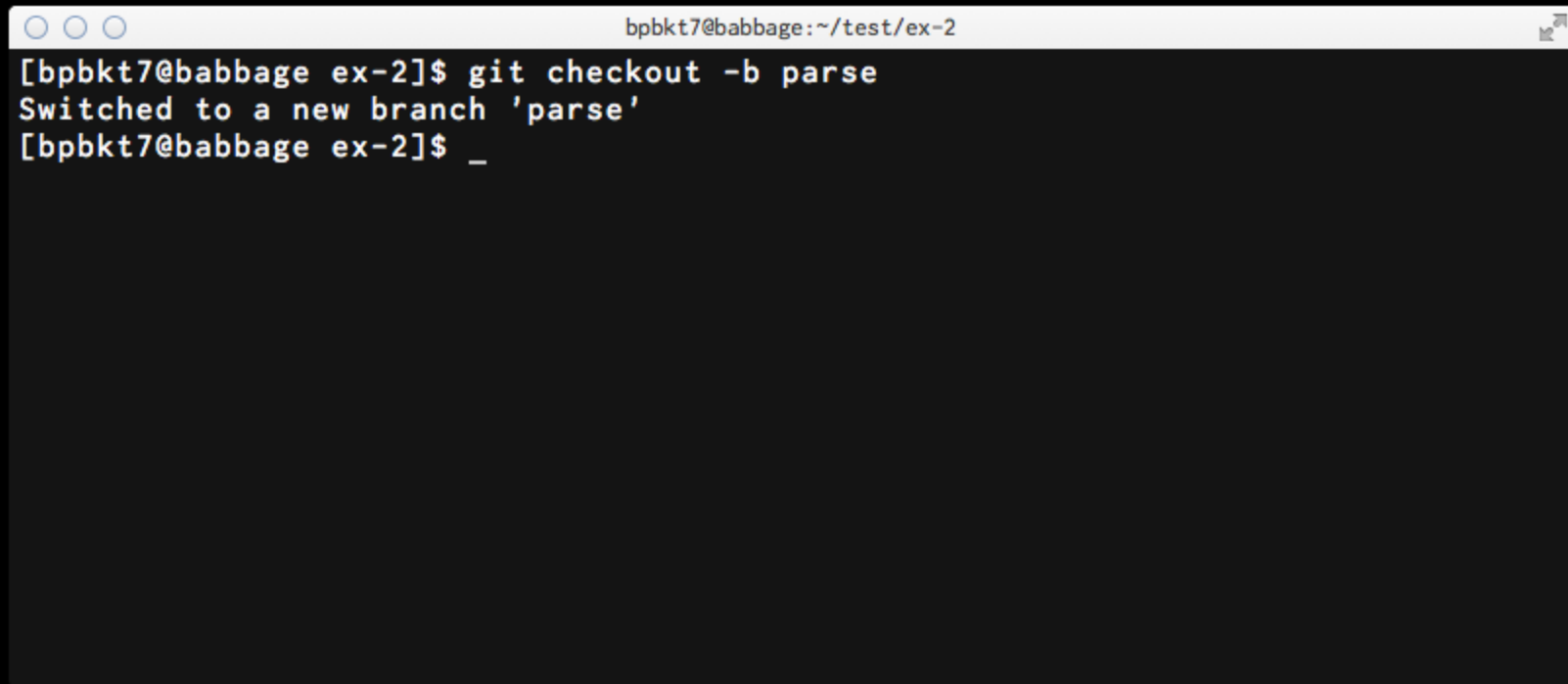
```
bpbkt7@babbage:~  
[bpbkt7@babbage ~]$ git clone git@github.com:bpbkt7/ex-2.git  
Initialized empty Git repository in /students/7/bpbkt7/ex-2/.git/  
Enter passphrase for key '/students/7/bpbkt7/.ssh/id_rsa':  
remote: Counting objects: 3, done.  
remote: Total 3 (delta 0), reused 0 (delta 0)  
Receiving objects: 100% (3/3), done.  
[bpbkt7@babbage ~]$ _
```



A terminal window with a title bar containing three window control buttons (minimize, maximize, close) and the text "bpbkt7@babbage:~/test/ex-2". The terminal content shows a sequence of commands and their output:

```
[bpbkt7@babbage test]$ cd ex-2/  
[bpbkt7@babbage ex-2]$ ls  
README.md  
[bpbkt7@babbage ex-2]$ _
```

```
bpbkt7@babbage:~/test/ex-2
[bpbkt7@babbage ex-2]$ touch parser.c
[bpbkt7@babbage ex-2]$ ls
parser.c  README.md
[bpbkt7@babbage ex-2]$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       parser.c
nothing added to commit but untracked files present (use "git add" to track)
[bpbkt7@babbage ex-2]$ _
```

A terminal window with a light gray title bar and three window control buttons (red, yellow, green) on the left. The title bar text is "bpbkt7@babbage:~/test/ex-2". The terminal content shows a command being executed and its output.

```
bpbkt7@babbage:~/test/ex-2
[bpbkt7@babbage ex-2]$ git checkout -b parse
Switched to a new branch 'parse'
[bpbkt7@babbage ex-2]$ _
```



bpbkt7@babbage:~/test/ex-2



```
[bpbkt7@babbage ex-2]$ git checkout -b parse
```

```
Switched to a new branch 'parse'
```

```
[bpbkt7@babbage ex-2]$ _
```



bpbkt7@babbage:~/test/ex-2



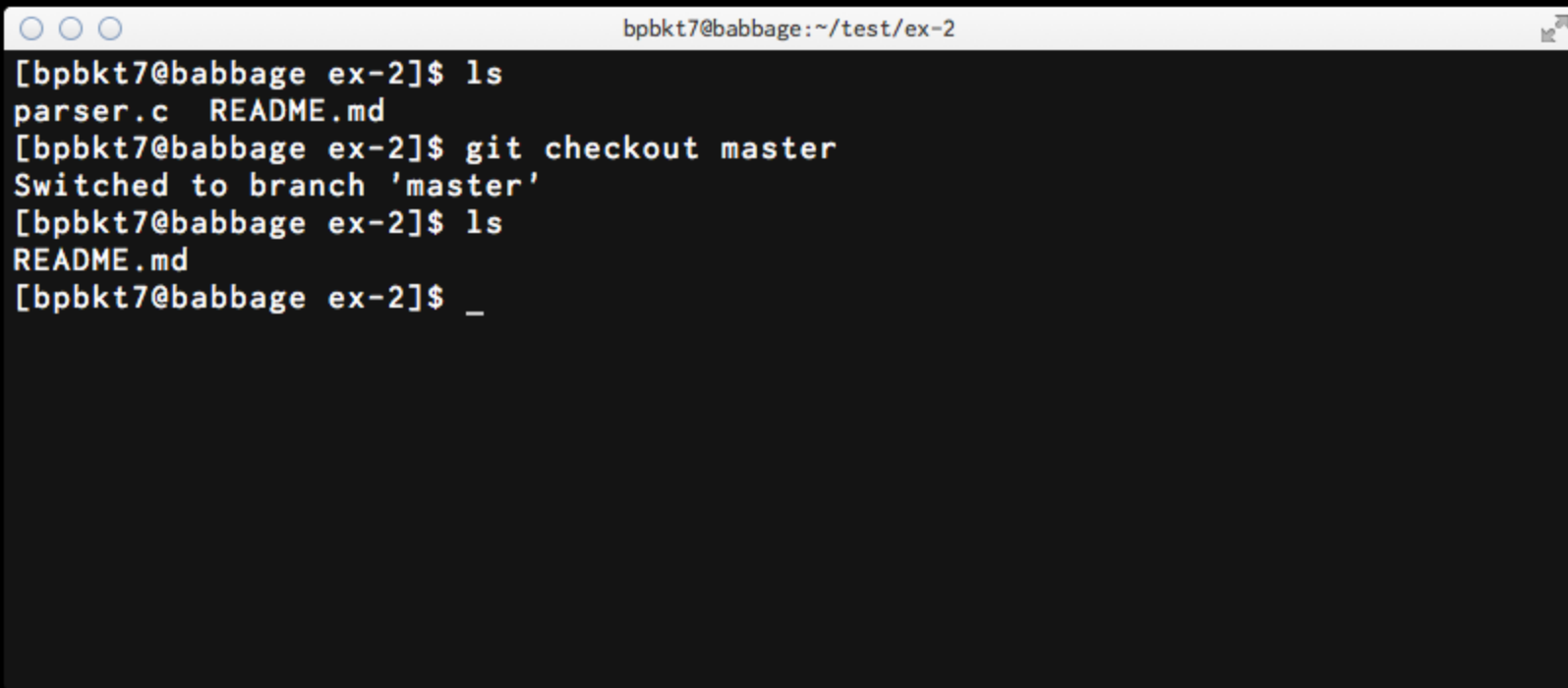
```
[bpbkt7@babbage ex-2]$ git status
# On branch parse
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       parser.c
nothing added to commit but untracked files present (use "git add" to track)
[bpbkt7@babbage ex-2]$ _
```

```

[bpbkt7@babbage ex-2]$ git add parser.c
[bpbkt7@babbage ex-2]$ git status
# On branch parse
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   parser.c
#
[bpbkt7@babbage ex-2]$ _
```

```
○ ○ ○ bpbkt7@babbage:~/test/ex-2
[bpbkt7@babbage ex-2]$ git add parser.c
[bpbkt7@babbage ex-2]$ git status
# On branch parse
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   parser.c
#
[bpbkt7@babbage ex-2]$ git commit -am "added parser"
[parse 052b960] added parser
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 parser.c
[bpbkt7@babbage ex-2]$ _
```



A terminal window with a title bar containing three window control buttons on the left and the text "bpbkt7@babbage:~/test/ex-2" on the right. The terminal content shows a sequence of commands and their outputs: "ls" lists "parser.c" and "README.md"; "git checkout master" switches to the master branch; a second "ls" shows only "README.md"; and the prompt ends with an underscore.

```
bpbkt7@babbage:~/test/ex-2
[bpbkt7@babbage ex-2]$ ls
parser.c  README.md
[bpbkt7@babbage ex-2]$ git checkout master
Switched to branch 'master'
[bpbkt7@babbage ex-2]$ ls
README.md
[bpbkt7@babbage ex-2]$ _
```



bpbkt7@babbage:~/test/ex-2



```
[bpbkt7@babbage ex-2]$ ls
parser.c  README.md
[bpbkt7@babbage ex-2]$ git checkout master
Switched to branch 'master'
[bpbkt7@babbage ex-2]$ ls
README.md
[bpbkt7@babbage ex-2]$ git pull
Enter passphrase for key '/students/7/bpbkt7/.ssh/id_rsa':
remote: Counting objects: 5, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From github.com:bpbkt7/ex-2
   0776dd2..4a3058f  master    -> origin/master
Updating 0776dd2..4a3058f
Fast-forward
 README.md |      5 +----
 1 files changed, 1 insertions(+), 4 deletions(-)
[bpbkt7@babbage ex-2]$ _
```



bpbkt7@babbage:~/test/ex-2



```
[bpbkt7@babbage ex-2]$ git merge parse
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 parser.c
[bpbkt7@babbage ex-2]$ ls
parser.c  README.md
[bpbkt7@babbage ex-2]$ _
```



bpbkt7@babbage:~/test/ex-2



```
[bpbkt7@babbage ex-2]$ git push_
```

Design your dev habits to  
minimize merge conflicts.

# merge conflicts

- Changes on the same file
- Different branches.
- Git handles it gracefully.

# fixing merge conflicts

- `git mergetool`
- `git checkout —theirs`
- `git checkout —ours`

# fixing merge conflicts

```
# # You have unmerged paths.  
# # (fix conflicts and run "git commit")  
# #  
# # Unmerged paths:  
# # (use "git add ..." to mark resolution)  
# #  
# # both modified:    README.md  
# #  
# no changes added to commit (use "git add" and/or "git commit -a")
```



# fixing merge conflicts

<<<<<<< HEAD

We like cats.

=====

We like dogs.

>>>>>>> branch-a

We like cats & dogs.

# fixing merge conflicts

We like cats & dogs.

fixing merge conflicts

add & commit

- Split up your development to avoid merge conflicts.
- Don't commit a broken build.
- Work in branches.
- Comment your commits.

```
git checkout -b myBranch
```

(make your changes)

```
git checkout master
```

```
git pull origin master
```

```
git merge myBranch
```

(fix conflicts if they exist)

```
git push origin master
```