

Module 4: Advance Git Commands



Module Objectives

At the end of this module, you will be able to:

- Identify the different Git commands.
- Use some of the key advance Git commands



Topic List

Key Git Commands

Advance Git Commands

Topic List

Key Git Commands

Advance Git Commands

Key Git Commands (1)

List of Key Git Commands

The following table lists the key Git commands and its description:

Creating a work space	
clone	Clone a repository into a new directory
init	Create an empty Git repository or reinitialize an existing one
Editing	
add	Add file contents to the index
mv	Move or rename a file, a directory, or a symlink
reset	Reset current HEAD to the specified state
rm	Remove files from the working tree and from the index

Analyzing the history and state	
bisect	Use binary search to find the commit that introduced a bug
grep	Print lines matching a pattern
log	Show commit logs
show	Show various types of objects
status	Show the working tree status



Key Git Commands (2)

List of Key Git Commands

The following table lists the key Git commands and its description:

Grow, mark and tweak your common history	
branch	List, create, or delete branches
checkout	Switch branches or restore working tree files
commit	Record changes to the repository
diff	Show changes between commits, commit and working tree, etc
merge	Join two or more development histories together
rebase	Reapply commits on top of another base tip
tag	Create, list, delete or verify a tag object signed with GPG
Collaborate	
fetch	Download objects and references from another repository
pull	Fetch from and integrate with another repository or a local branch
push	Update remote references along with associated objects



Topic List

Key Git Commands

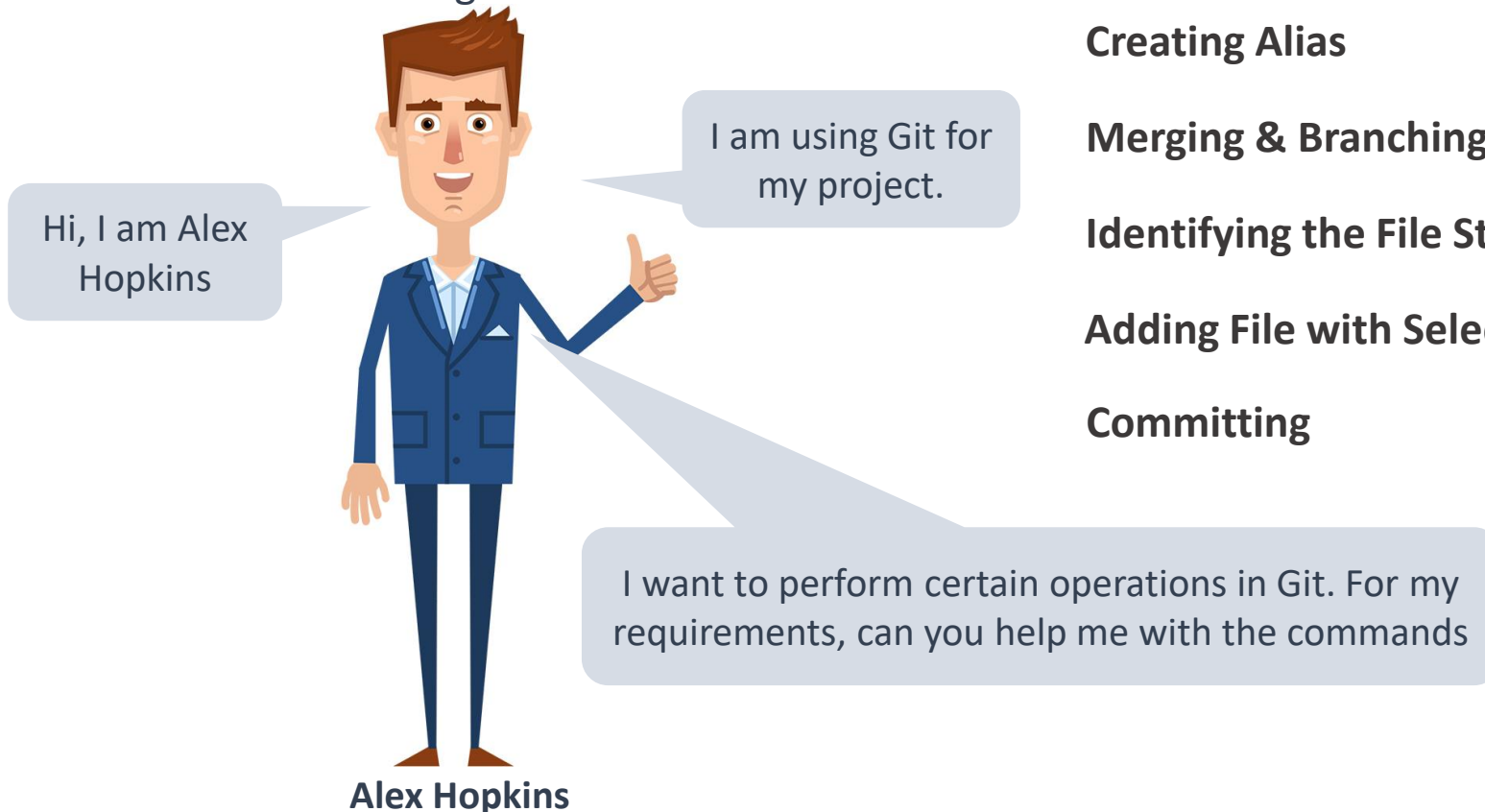
Advance Git Commands

Advance Git Commands

Introduction

Scenario

In this module, requirements of Alex Hopkins are captured and solutions are provided for his requirement. Along with Alex you will also learn the usage of advance Git commands:



Creating Alias

Merging & Branching

Identifying the File Status

Adding File with Selected Changes

Committing

Rebasing

Resetting the files

Finding Difference

Stashing

Parameters for better logging

Cleaning untracked files and directories

Git Bisect & Blameon



Creating Alias

Scenario

Alex wants to create an alias for most frequently used command based on the history of commit. He creates an alias by using the following command to save his writing time.

```
git config --global alias.XZ
```

Example

```
git config --global alias.X "log --oneline --graph"
```

After creating alias the command can be as follows.

```
git X instead of git log --oneline --graph.
```

```
If more parameters to the alias then git X --author="Charles"
```



Merging & Branching (1)

Scenario

Alex developed some code in his local branch. He wants to merge the changes to another branch. He wants to keep the Git branch history during a pull request, even after merging his local branch with another.

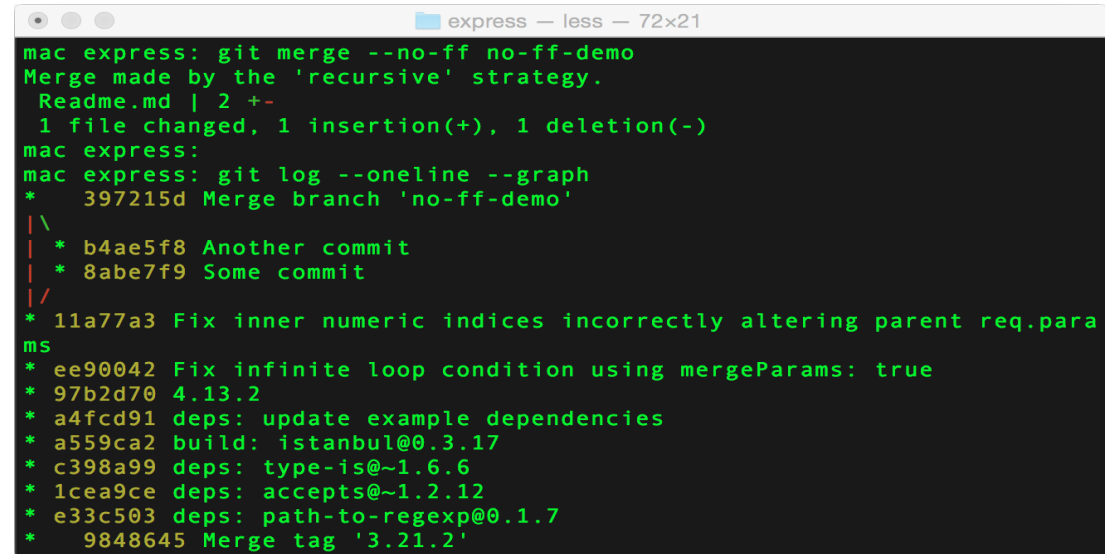
Merge Command

Merge command joins two or more development histories together. To merge the histories together, go to the branch to which you need to merge and use the following command:

```
git merge some-branch-name
```

The following command helps to store a commit history tree where the branch changes are tracked.

```
git merge --no-ff  
git merge --no-ff some-branch-name.
```



```
mac express: git merge --no-ff no-ff-demo  
Merge made by the 'recursive' strategy.  
  README.md | 2 +-  
  1 file changed, 1 insertion(+), 1 deletion(-)  
mac express:  
mac express: git log --oneline --graph  
* 397215d Merge branch 'no-ff-demo'  
| \  
| * b4ae5f8 Another commit  
| * 8abe7f9 Some commit  
| /  
* 11a77a3 Fix inner numeric indices incorrectly altering parent req.para  
ms  
* ee90042 Fix infinite loop condition using mergeParams: true  
* 97b2d70 4.13.2  
* a4fcd91 deps: update example dependencies  
* a559ca2 build: istanbul@0.3.17  
* c398a99 deps: type-is@~1.6.6  
* 1cea9ce deps: accepts@~1.2.12  
* e33c503 deps: path-to-regexp@0.1.7  
* 9848645 Merge tag '3.21.2'
```

Alex needs to remember the branch structure after a local merge which is difficult.
How will Alex know the branching details?



Merging & Branching (2)

Branch Command

To view all the remote branches, use the following command:

```
git branch -a
```

To display branches which are fully merged with the master branch, you can use the following command:

```
git branch -a --merged
```

```
mac express: git branch -a
* master
remotes/origin/1.x
remotes/origin/2.x
remotes/origin/3.x
remotes/origin/4.x
remotes/origin/5.0
remotes/origin/5.x
remotes/origin/HEAD -> origin/master
remotes/origin/benchmark
remotes/origin/external-isAbsolute
remotes/origin/master
remotes/origin/mscdex-router-optimize
remotes/origin/slim-benchmark
remotes/origin/streaming-render
mac express: git branch -a --merged
* master
remotes/origin/3.x
remotes/origin/4.x
mac express: 
```



Merging & Branching (3)

Sort Command

Use the following command for viewing the remote branches, that exists along with the last commits.

```
git for-each-ref --sort=committerdate --format='%(refname:short) * %(authorname) * %(committerdate:relative)'  
refs/remotes/ | column -t -s '*'
```

```
mac express: git for-each-ref --sort=committerdate --format='%(refname:short) * %(authorname) * %(committerdate:relative)' refs/remotes/ | column -t -s '*'  
origin/1.x                Tj Holowaychuk          4 years, 6 months ago  
origin/2.x                TJ Holowaychuk          3 years ago  
origin/streaming-render   Roman Shtylman          1 year, 10 months ago  
origin/benchmark          Douglas Christopher Wilson 1 year, 2 months ago  
origin/mscdex-router-optimize Douglas Christopher Wilson 1 year, 1 month ago  
origin/slim-benchmark     Douglas Christopher Wilson 1 year, 1 month ago  
origin/5.x                Douglas Christopher Wilson 10 months ago  
origin/external-isAbsolute Jeremiah Senkpiel        5 months ago  
origin/4.x                Douglas Christopher Wilson 9 weeks ago  
origin/5.0                Douglas Christopher Wilson 9 weeks ago  
origin/3.x                Douglas Christopher Wilson 5 weeks ago  
origin/master             Brendan Ashworth        5 weeks ago  
origin/HEAD               Brendan Ashworth        5 weeks ago  
mac express: □
```

Alex has the details about the remote branches. How can he extract files from those remote branches?



Merging & Branching (4)

Extracting Files from Branches

To extract files from other branches, use the following command:

```
git show some-branch:file.js
```

To redirect the output to a temporary file, which can be deleted later, use the following command:

```
git show some-branch-name:file.js > deleteme.js
```



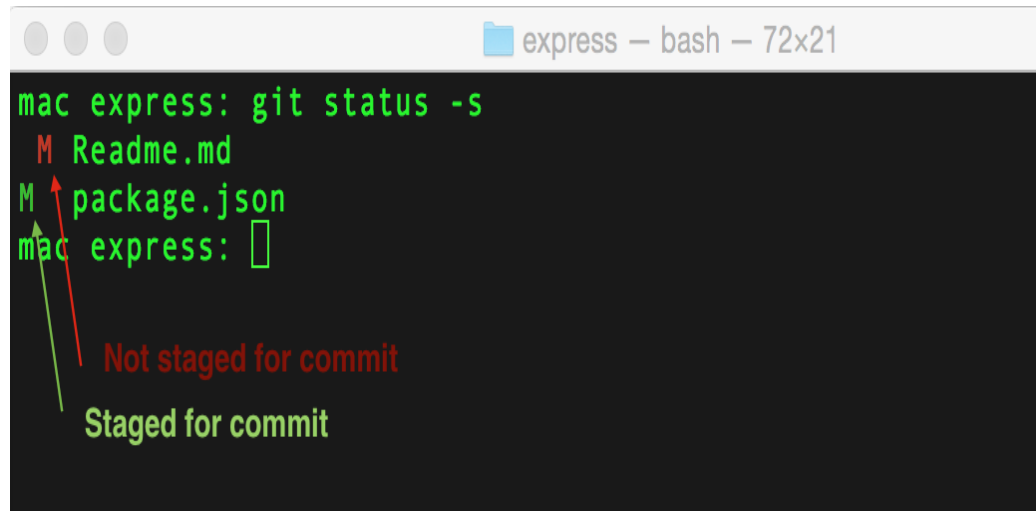
Identifying the File Status

Scenario

Alex has his local files with similar names. He has forgotten what was the last action taken on those files. He wants to display which stage is a file in Git.

To identify the file status, use the following command:

```
git status -s
```



```
mac express: git status -s
 M README.md
 M package.json
mac express: 
```

The terminal window shows the output of the `git status -s` command. The first two lines are `M README.md` and `M package.json`. A red arrow points from the text "Not staged for commit" to the `M` in the first line. A green arrow points from the text "Staged for commit" to the `M` in the second line.

Remember

Following are the difference status of the file:

1. Yet to be staged for commit
2. Staged for commit
3. Committed



Adding File with Selected Changes

Scenario

Alex finds a file which is yet to be staged, however in that file, he wants to add only selected changes from a file.

To add a file with selected changes, use the following command:

```
git add -p
```

```
mac express: git add -p
diff --git a/package.json b/package.json
index 3b4389e..5112b4d 100644
--- a/package.json
+++ b/package.json
@@ -1,5 +1,5 @@
 {
-  "name": "express",
+  "name": "express-custom",
   "description": "Fast, unopinionated, minimalist web framework",
   "version": "4.13.2",
   "author": "TJ Holowaychuk <tj@vision-media.ca>",
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
@@ -1,5 +1,5 @@
 {
-  "name": "express",
+  "name": "express-custom",
   "description": "Fast, unopinionated, minimalist web framework",
   "version": "4.13.2",
   "author": "TJ Holowaychuk <tj@vision-media.ca>",
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? 
```



Commit (1)

Scenario

Alex added and committed the selected changes. However, he realizes that there is a typo after committing. He wants to correct the typo in his previous commit, instead of making a new commit with comments as “typo”.

When the changes are not pushed to the remote branch, you can use amend option:

- 1) Stage the fixed file using the following command:

```
git add corrected-file.js
```

- 2) Execute the following “amend” command through which the most recent changes can be added to the latest commit. Even the commit message can be edited.

```
git commit --amend
```

Important

It is recommended to avoid a new commit, instead of fixing an old commit.

How will Alex fix the typo error if the changes are already **Pushed** to remote branch?



Commit (2)

Scenario

When the changes are pushed to the remote branch. You can use force option to fix the incorrect commit.

The following command is used to fix the incorrect commit:

```
git push -f
```

Note: When you are performing this task, you would be changing the content on a branch that is being **used by other users**.



Commit (3)

Standards for Commit Message

A commit should describe the following sentence:

```
When applied, this commit will: {{ YOUR COMMIT MESSAGE }}
```

Example

- When applied this commit will **Update README file**
- When applied this commit will **Add validation for GET /user/:id API call**
- When applied this commit will **Revert commit 12345**

Soft revert on a commit options are available at

This reverts now and re-commit reverted files and takes the new commit message as an input.

```
git revert -n
```



Commit (4)

cherry Command

The git cherry command finds the commits where are yet to be applied to upstream.

```
git cherry-pick -v SampleBranchName  
Git cherry -v master
```

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)  
$ git cherry -v master  
+ 0aa886c2d9c80ddadc659182935eadd7949d4b31 Adding sample project contents
```

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)  
$ git cherry -v develop  
  
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)  
$
```

cherry-pick Command

The git cherry-pick command applies the changes introduced by some existing commits.

```
git cherry-pick <commitSHA>
```



Rebasing (1)

Scenario

Alex commits in a local version of a master branch. Simultaneously, another user Tom checked in 5 days of work into the master branch.

By using Push & Pull command, the scenario can be handled

- When the user attempts to push, Git identifies the conflicts and it prompts to perform Git pull to resolve it.
- After doing Git pull, Git throws a message “Merge remote-tracking branch ‘origin/master’”, which might clutter the log history a bit.

Using Rebase Command

Alternatively, to address the considered scenario, you can use the following command:

```
git pull --rebase
```

- The rebase command forces Git to pull the changes first, and then re-apply (rebase) the un-pushed commits on the remote branch’s latest version. Thus the merge and the message is avoided.



Rebasing (3)

Merging Branch with Master

- Rebase command is an alternative to Merge command. You can rebase the feature branch with master branch using the following commands:

```
git checkout feature  
git rebase master
```

- These commands move the entire feature branch to begin from the master branch and it incorporate all new commits in master.
- Rebasing re-writes the project history by creating brand new commits for each commit in the original branch.
- The major benefit of rebasing is:
 - Cleaner project history
 - Elimination of unnecessary merge commits required by git merge command
 - Provides linear project history
 - Easier navigation in the project



Rebasing (2)

Interactive Rebasing

If you are in need of an interactive rebasing, use the following command:

```
git rebase -i {{some commit hash}}
```

```
git pull --rebase
```

The command reapplies commit on top of another base.

```
.g/r/git-rebase-todo
pick afece96 Commit 1
pick a6671aa Commit 2
pick 2261d76 Commit 3
pick 8f462e7 Commit 4
pick 7c9d609 First bad commit
pick e549232 Commit 5
pick e5efc0e Commit 6

# Rebase 11a77a3..e5efc0e onto 11a77a3
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
<rebase-merge/git-rebase-todo CWD: /Users/alexk/Desktop/express Line: 1
```

Note: It is recommended not to use rebase commits that exist outside the repository.



Resetting the Files (1)

Types of Resets

You can use the Reset command to return to a particular version in Git history. Reset command lets you tidy up code before doing commit

Command	Description
git reset --hard {{some-commit-hash}}	Changes made after the commit are discarded.
git reset {{some-commit-hash}}	Changes made after commit are moved to “ not yet staged for commit ” stage. If needed, run Git add and Git commit to add the files back to the repository.
git reset --soft {{some-commit-hash}}	Changes made after the commit are moved to “ staged for commit ” stage. You can run Git commit command to add files back to the repository.



Resetting the Files (2)

Scenario

Alex wants to Ignore local changes for few files, but at the same time he wants to save the changes in few other files. He wants to check out committed versions of the files whose changes are to be ignored.

```
git checkout filename.js
```

An alternative command is:

```
git reset --hard
```

Note: But this option is only for some files.

To check out another version of a file from other branch or commit.

```
git checkout some-branch-name file-name.js
```

```
git checkout {{some-commit-hash}} file-name.js
```

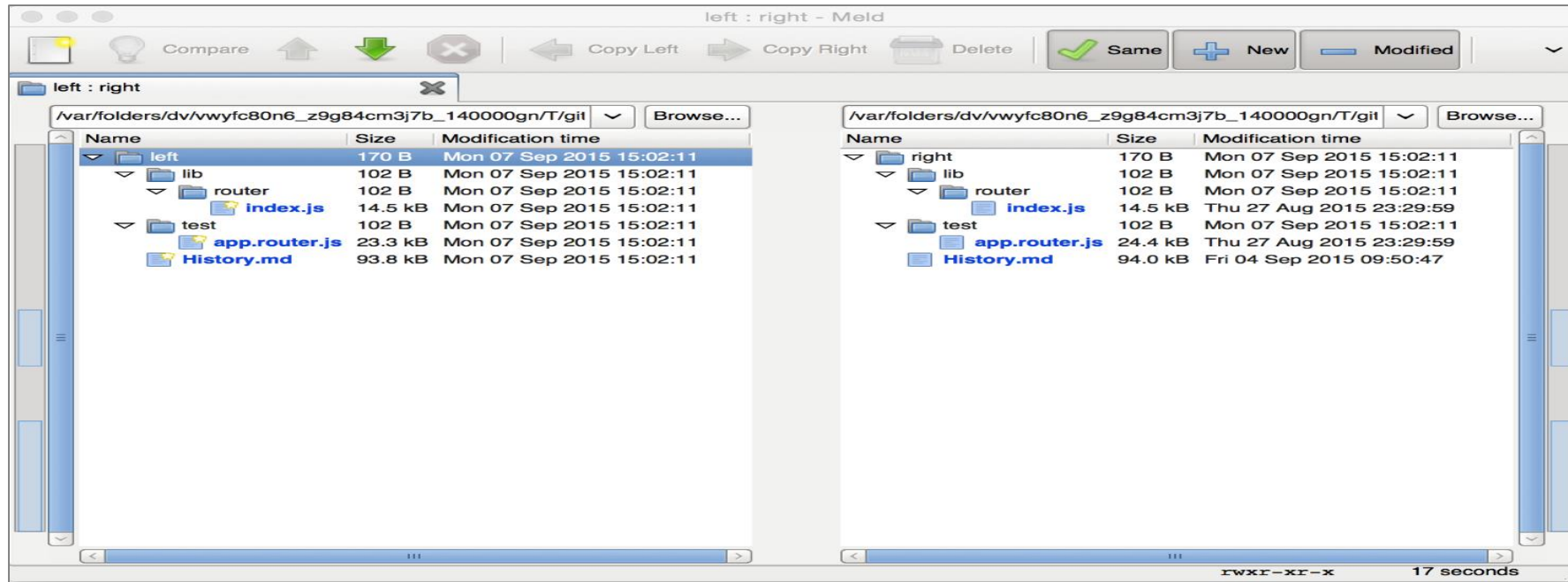


Finding Difference Between Folders (1)

Scenario

Alex wants to find the difference between the folders. You can use the “difftool” command to find the difference between folders.

```
git difftool -d
```



Finding Difference Between Folders (2)

Finding Difference Between Files

To find the difference between two files, use the following command:

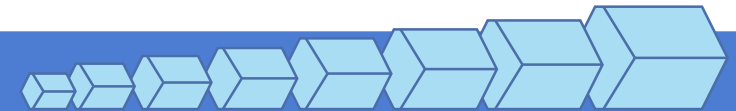
```
git diff some-branch some-filename.js
```

Alex wants to ignore the white space during re-indentation or re-format of a file. How will he do this?

Ignore the white space

To ignore the white space changes during re-indentation or re-format of a file, use the following command:

```
git diff -w or git blame -w
```



Stashing (1)

Scenario

Alex wants to pull all the unsaved changes on Git Stack.

You can put all the unsaved changes on a “Git stack” using the “stash” command.

```
git stash -p
```

-p option allows you to see the chunks of changes that needs to be stashed.

You can use the stash command when you want to record the current state of the working directory and the index, but want to go back to a clean working directory.

The command saves the local modifications and reverts the working directory to match the HEAD commit.

```
mac express: git stash -p
diff --git a/Readme.md b/Readme.md
index 8da83a5..4735255 100644
--- a/Readme.md
+++ b/Readme.md
@@ -1,6 +1,6 @@
[![Express Logo](https://i.cloudup.com/zfY6l7eFa-3000x3000.png)](http://expressjs.c
om/)

- Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..
+ Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..

[![NPM Version][npm-image]][npm-url]
[![NPM Downloads][downloads-image]][downloads-url]
Stash this hunk [y,n,q,a,d,/,e,?]? ?
y - stash this hunk
n - do not stash this hunk
q - quit; do not stash this hunk or any of the remaining ones
a - stash this hunk and all later hunks in the file
d - do not stash this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
@@ -1,6 +1,6 @@
[![Express Logo](https://i.cloudup.com/zfY6l7eFa-3000x3000.png)](http://expressjs.c
om/)

- Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..
+ Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..

[![NPM Version][npm-image]][npm-url]
[![NPM Downloads][downloads-image]][downloads-url]
Stash this hunk [y,n,q,a,d,/,e,?]? [y]
```



Stashing (2)

Stash Command Options

Stash list

```
git stash list
```

- Lists the current stash entries that you have. List command displays the following:
 - A named stash entry (e.g. `stash@{0}` which will be the latest entry, `stash@{1}` is the one before, etc.),
 - A short description of the commit the entry was based on.

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject/spring-petclinic/src/main/resources/messages (develop)
$ git stash list
stash@{0}: WIP on develop: 534f9ec Merge pull request #4 from RobertNorthard/feature/dependency-updates
```

Stash pop

```
git stash pop
```

- Removes a single stashed state from the stash list and apply it on top of the current working tree state.

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject/spring-petclinic/src/main/resources/messages (develop)
$ git stash pop
Auto-merging src/main/resources/messages/messages.properties
CONFLICT (content): Merge conflict in src/main/resources/messages/messages.properties
```



Parameters for Logging (1)

Scenario

Alex needs a weekly report about the project files which has to be submitted every Friday.
On Fridays, he needs to execute following command:

```
git log --author="Alex Hopkins" --after="1 week ago" --oneline
```

Following table lists the parameters that can be passed on with the log command:

Parameters	Description
author="Alex Hopkins"	Show commits made by a particular author
name-only	Only file names are displayed.
oneline	Data is compressed to one line.
graph	Displays the tree of dependency for all commits.
reverse	Commits in reverse order (Oldest commit first).
after	Filters and displays commits done after particular date.
before	Displays commits before particular date.

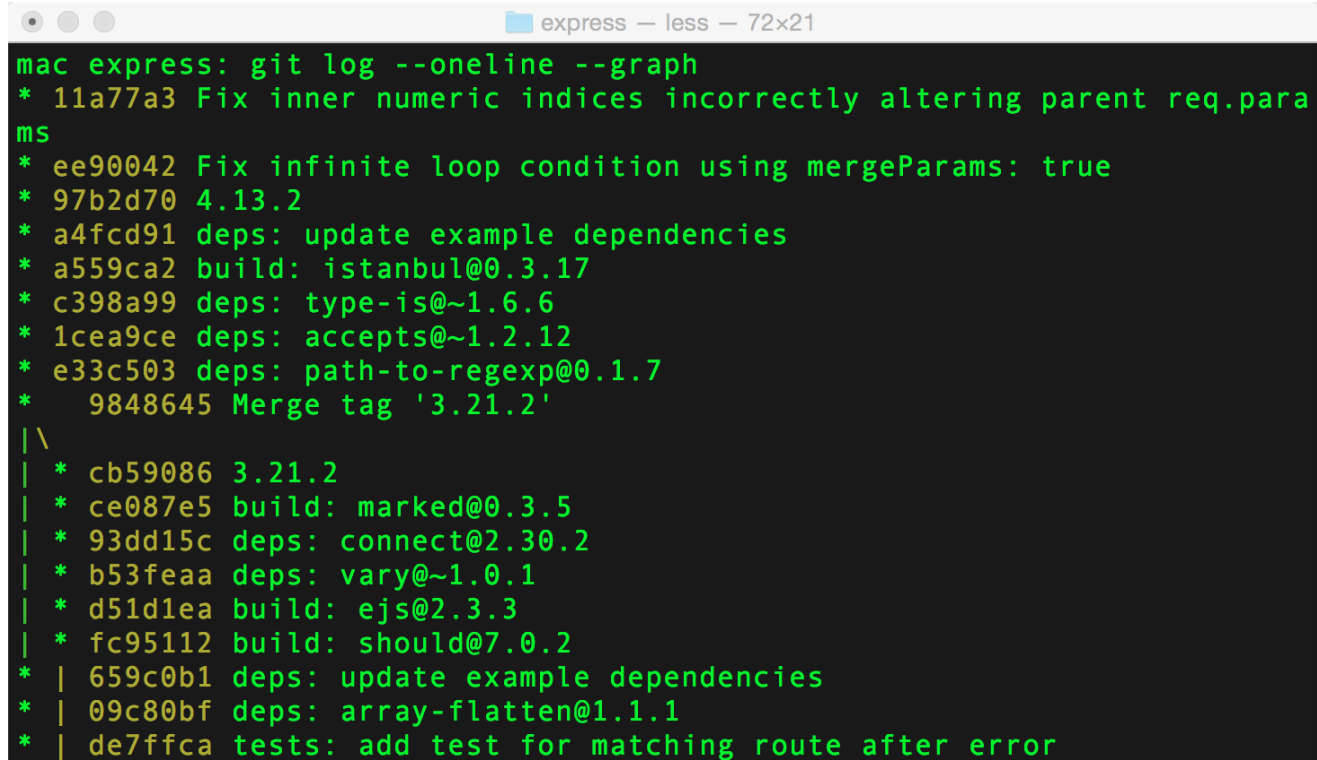


Parameters for Logging (2)

Example

Alex wants to have a look at the compressed data as tree of dependency for all commits.

```
git log --oneline --graph
```



```
mac express: git log --oneline --graph
* 11a77a3 Fix inner numeric indices incorrectly altering parent req.params
* ee90042 Fix infinite loop condition using mergeParams: true
* 97b2d70 4.13.2
* a4fcd91 deps: update example dependencies
* a559ca2 build: istanbul@0.3.17
* c398a99 deps: type-is@~1.6.6
* 1cea9ce deps: accepts@~1.2.12
* e33c503 deps: path-to-regexp@0.1.7
* 9848645 Merge tag '3.21.2'
| \
| * cb59086 3.21.2
| * ce087e5 build: marked@0.3.5
| * 93dd15c deps: connect@2.30.2
| * b53feaa deps: vary@~1.0.1
| * d51d1ea build: ejs@2.3.3
| * fc95112 build: should@7.0.2
* | 659c0b1 deps: update example dependencies
* | 09c80bf deps: array-flatten@1.1.1
* | de7ffca tests: add test for matching route after error
```



Parameters for Logging (3)

Example

Alex wants to have a look at all the changes that were made in each commit, along with the commit message, author, and date.

```
git log -p filename
```

```
express — less — 72x21
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date: Thu Oct 23 02:20:51 2014 -0400

docs: visionmedia is now tj on Github

commit 5f7a37ee517e172c0762fc3debaf94c066e531f9
Author: Fishrock123 <fishrock123@rocketmail.com>
Date: Sat Oct 11 14:12:03 2014 -0400

docs: misc. tweaks

closes #2394

commit ff3a368b2f9a7e640fb3fd18f116de5352e73d58
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date: Thu Oct 23 02:08:34 2014 -0400

deps: update example dependencies

commit ccc45a74f89b45a6551bc8ff305581dbc8175bca
/visionmedia
```



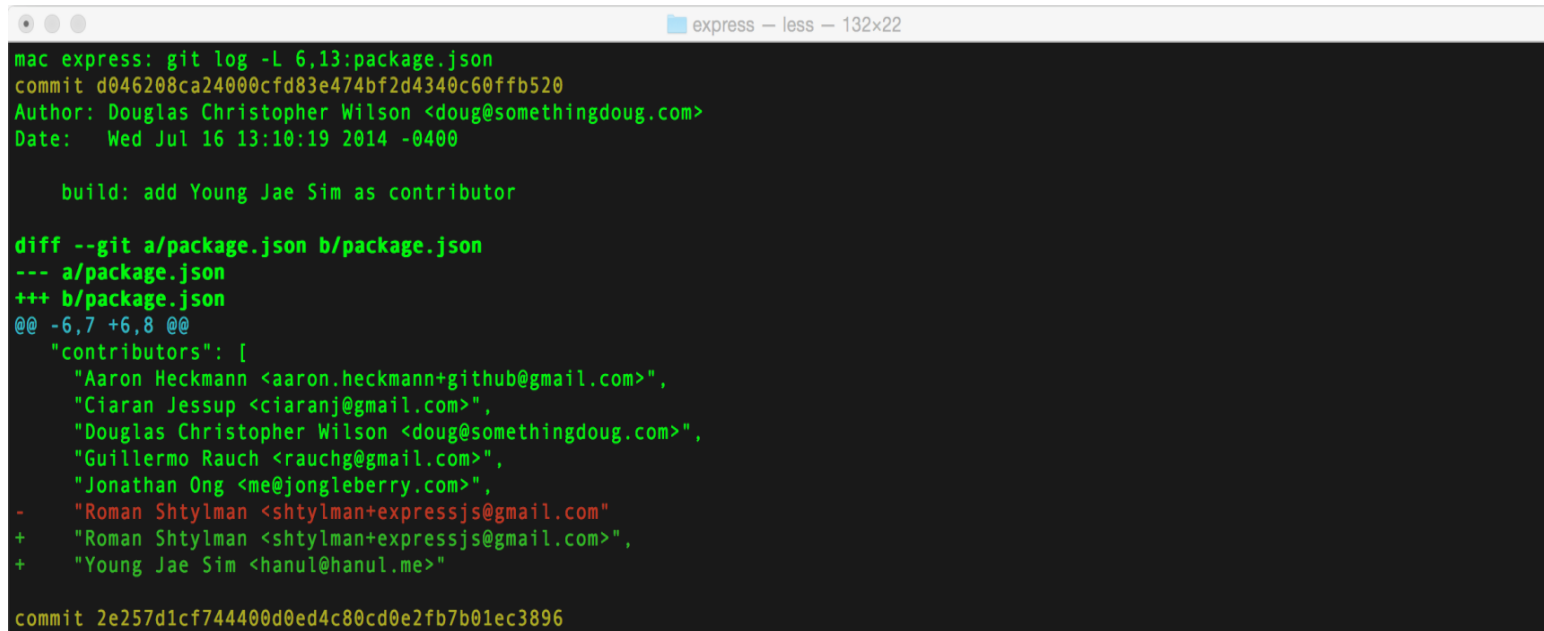
Parameters for Logging (4)

Example

Alex finds something fishy in line 1 of a certain file. He wants to see what has changed in it through various commits.

```
git log -L 1,1:filename
```

This traces the evolution of the line 1 to line 1 in the file named “filename” through various commits.

A terminal window titled 'express — less — 132x22' showing the output of the command 'git log -L 6,13:package.json'. The output shows two commits. The first commit, d046208ca24000cfd83e474bf2d4340c60ffb520, is by Douglas Christopher Wilson and dated Wed Jul 16 13:10:19 2014. It includes a build message 'add Young Jae Sim as contributor'. The second commit, 2e257d1cf744400d0ed4c80cd0e2fb7b01ec3896, is a diff between two versions of package.json. It shows the addition of 'Young Jae Sim' to the contributors list, replacing 'Roman Shtylman'.

```
mac express: git log -L 6,13:package.json
commit d046208ca24000cfd83e474bf2d4340c60ffb520
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date:   Wed Jul 16 13:10:19 2014 -0400

    build: add Young Jae Sim as contributor

diff --git a/package.json b/package.json
--- a/package.json
+++ b/package.json
@@ -6,7 +6,8 @@
  "contributors": [
    "Aaron Heckmann <aaron.heckmann+github@gmail.com>",
    "Ciaran Jessup <ciaranj@gmail.com>",
    "Douglas Christopher Wilson <doug@somethingdoug.com>",
    "Guillermo Rauch <rauchg@gmail.com>",
    "Jonathan Ong <me@jongleberry.com>",
-   "Roman Shtylman <shtylman+expressjs@gmail.com>",
+   "Roman Shtylman <shtylman+expressjs@gmail.com>",
+   "Young Jae Sim <hanul@hanul.me>"
  ]

commit 2e257d1cf744400d0ed4c80cd0e2fb7b01ec3896
```



Parameters for Logging (5)

Example

Alex wants to log the changes that are yet to be merged to the parent branch.

```
git log --no-merges master
```

no-merges flag: Displays changes which are not merged to any branch, and the master.

Alternatively,

```
git show --no-merges master
```

or

```
git log -p --no-merges master
```

Also displays file changes which have not been merged yet.



Cleaning Untracked Files and Directories

Scenario

Alex identifies few untracked files and directories in the project which he wants to remove.

Following are the commands to remove untracked files and directories:

Git clean -f

Removes untracked files

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$ git clean -f
Removing test.txt

rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$
```

Git clean -fd

Removes untracked files/directories

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$ git clean -fd
Removing Sample_Source_Code/
Removing test/

rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$
```

Git clean -nfd

Lists all files and folders which will be removed

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$ git clean -nfd
Would remove sample/
Would remove test/

rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$
```



Git Bisect & Blameon (1)



Alex went on a planned vacation.



On checking the last commit before vacation, the left out feature is intact and is operational.



Alex headed back to work. He pulled the latest changes of the feature that he worked prior to vacation and noticed that the work is left cracked.



- Numerous commits exist after he left for vacation.
- Found bugs in the feature that he was working.

Now, how will Alex fix the bug and find who broke the feature???



Git Bisect & Blameon (2)



Git Blameon & Git Bisect

Git Blameon command to find out the culprit who broke feature.

Git Bisect command to find the broken commit that caused the bug by using the divide and conquer algorithm.

```
$ git bisect
```

Git Bisect Conquer Algorithm

- A known bad commit and a known good commit are mentioned to Git Bisect.
- This splits the commits between those in half, thus identifies the middle one and checkout a new (nameless) branch.
- Alex should check whether the feature is broken in the middle commit.
- If the middle commit is operational and not broken, then repast Git bisect with this known good commit and the known bad commit.
- This process gets narrowed down and thus the first bad commit which broke the feature is spotted.



Git Bisect & Blameon (3)

Git Bisect Command Example

Following screen captures shows an example of Git Bisect command.

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$ git bisect
usage: git bisect [help|start|bad|good|new|old|terms|skip|next|reset|visualize|replay|log|run]
```

```
rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop)
$ git bisect good
You need to start by "git bisect start"
Do you want me to do it for you [Y/n]? Y

rakhi.parashar@M2B-L-5296SMP MINGW32 /c/Data/DevopsAcademy/ExampleProject (develop|BISECTING)
$
```



Knowledge Check

Select the right answer

Which command is used to find the broken commit that causes the bug?


- a) git log
- ☒ b) git bisect
- c) git blameon
- d) git checkout



Knowledge Check

Select the right answer

Which command saves the unsaved changes on a “Git stack”?

- a) Clone
- b) Add
- c) Commit
-  d) Stash



Exercise 4.1: Using Git Advance Commands

Scenario

- Create a branch devTest, pull the changes from the master branch over here and then make some more changes, save those changes and then commit your changes. Go to develop branch and then merge the changes of devTest branch in develop branch.
- Compare both the branches develop and devTest with master, if any file is missing from the master then fetch that file and place in both branches (develop and devTest). Use commands like grep, log, show, status, diff and bisect for analyzing history.
- Add a new file in the working tree and make the changes and commit. Post this copy this file to another file File2 and then rename the newly added file. Finally remove the File2 from the working tree. Check status of the working tree.
- Clone the spring petclinic project from git repository. Go to spring-petclinic/src/main/resources/messages/messages.properties and remove the first line with content "welcome=Welcome". OR you can create any other bug and commit the changes. Now use binary search to find the commit that introduced a bug. Hint: use Bisect and blame command.



Exercise 4.2: Using Git Advance Commands

Scenario

Following are the key scenarios where you need to use some of the other Git Advance Commands:

- Creating an alias for most frequently used command based on the history of commit.
- Identifying the file status by displaying which stage is a file in Git
- Merging without using merge command
- Keeping the changes in local branch that you want to commit later.
- Committing only specific commits from master to your current
- Removing untracked files and directories



Module Summary

Now, you should be able to:

- Identify the different Git commands.
- Use some of the key advance Git commands



Course Summary

Now, you should be able to:

- Describe Configuration Management (CM).
- Explain what is Git.
- Practice the steps to create Git Hub Repository.
- Use some of the key Git commands for version control.



Thank You