Git Essentials

LEARNING VOYAGE

Table of Contents

1. Getting Started with Git

Getting Started with Git

- In this first lesson, we will start at the very beginning, assuming that you do not have Git on your machine.
- This course is intended for developers who have never used Git or only used it a little bit, but who are scared to throw themselves headlong into it.
- If you have never installed Git, this is your lesson.
- If you already have a working Git box, you can quickly read through it to check whether everything is alright.

Installing Git

- Git is open source software.
- You can download it for free from http://git-scm.com, where you will find a package for all the most common environments (GNU-Linux, macOS and Windows).
- At the time of writing this course, the latest version of Git is 2.11.0.

Installing Git on GNU-Linux

nando : bash — Konsole

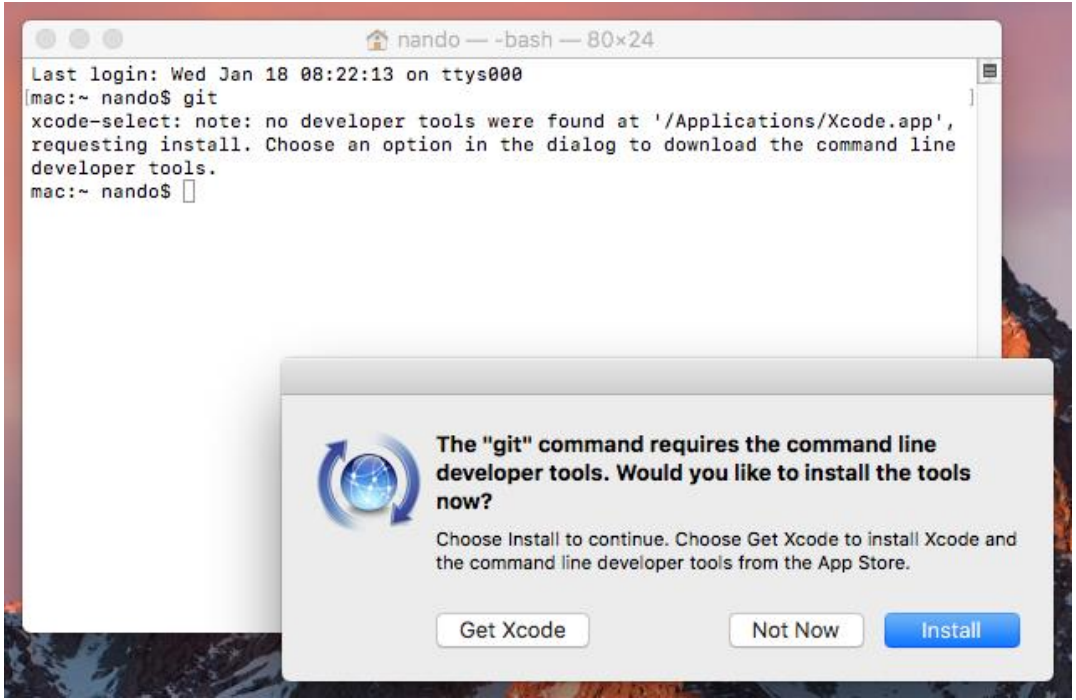
File Edit View Bookmarks Settings Help

```
nando@kubu:~$ git
The program 'git' is currently not installed. You can install it by typing:
sudo apt install git
nando@kubu:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.4.0-31 linux-headers-4.4.0-31-generic linux-image-4.4.0-31-generic linux-image-extra-4.4.0-31-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-arch git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 3 not upgraded.
Need to get 3.760 kB of archives.
After this operation, 25,6 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://it.archive.ubuntu.com/ubuntu xenial/main amd64 liberror-perl all 0.17-1.2 [19,6 kB]
Get:2 http://it.archive.ubuntu.com/ubuntu xenial/main amd64 git-man all 1:2.7.4-0ubuntu1 [735 kB]
Get:3 http://it.archive.ubuntu.com/ubuntu xenial/main amd64 git amd64 1:2.7.4-0ubuntu1 [3.006 kB]
Fetched 3.760 kB in 7s (509 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 223060 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17-1.2_all.deb ...
Unpacking liberror-perl (0.17-1.2) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.7.4-0ubuntu1_all.deb ...
Unpacking git-man (1:2.7.4-0ubuntu1) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.7.4-0ubuntu1_amd64.deb ...
Unpacking git (1:2.7.4-0ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up liberror-perl (0.17-1.2) ...
Setting up git-man (1:2.7.4-0ubuntu1) ...
Setting up git (1:2.7.4-0ubuntu1) ...
nando@kubu:~$ git --version
git version 2.7.4
nando@kubu:~$
```

nando : bash

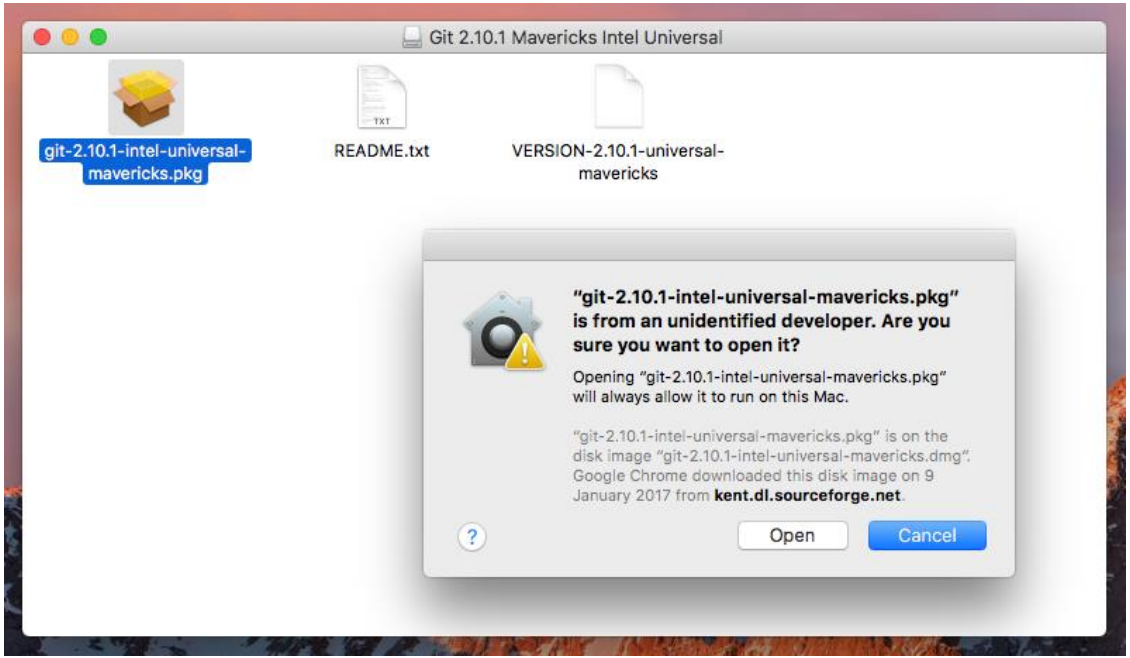
nando : bash — Konsole 13:19

Installing Git on macOS



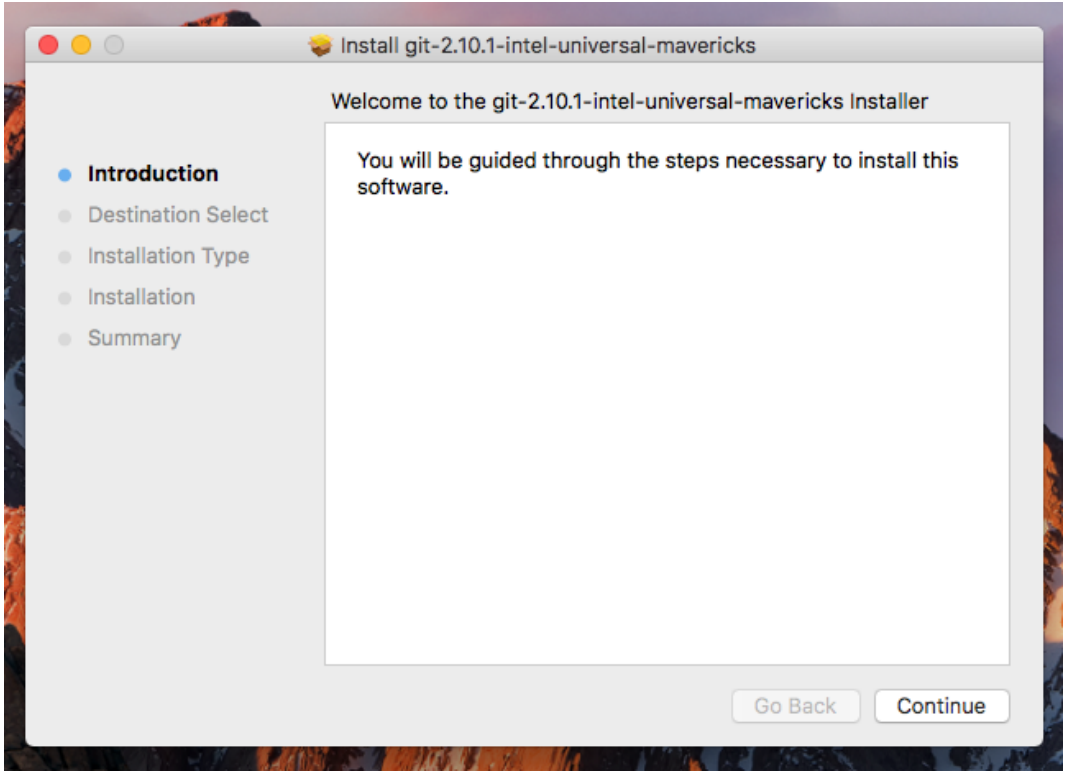
Clicking on the Install button will fire the installation process.

Installing Git on macOS



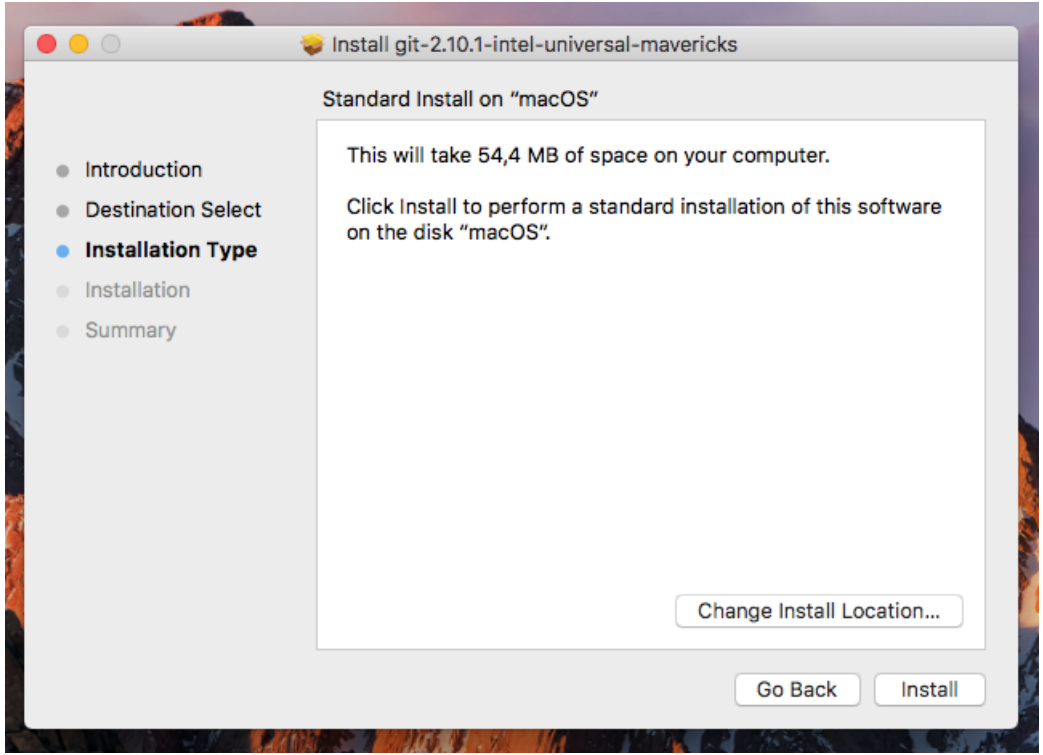
Hold down CTRL and click to let macOS prompt you to open the package

Installing Git on macOS



Let's start the installation process

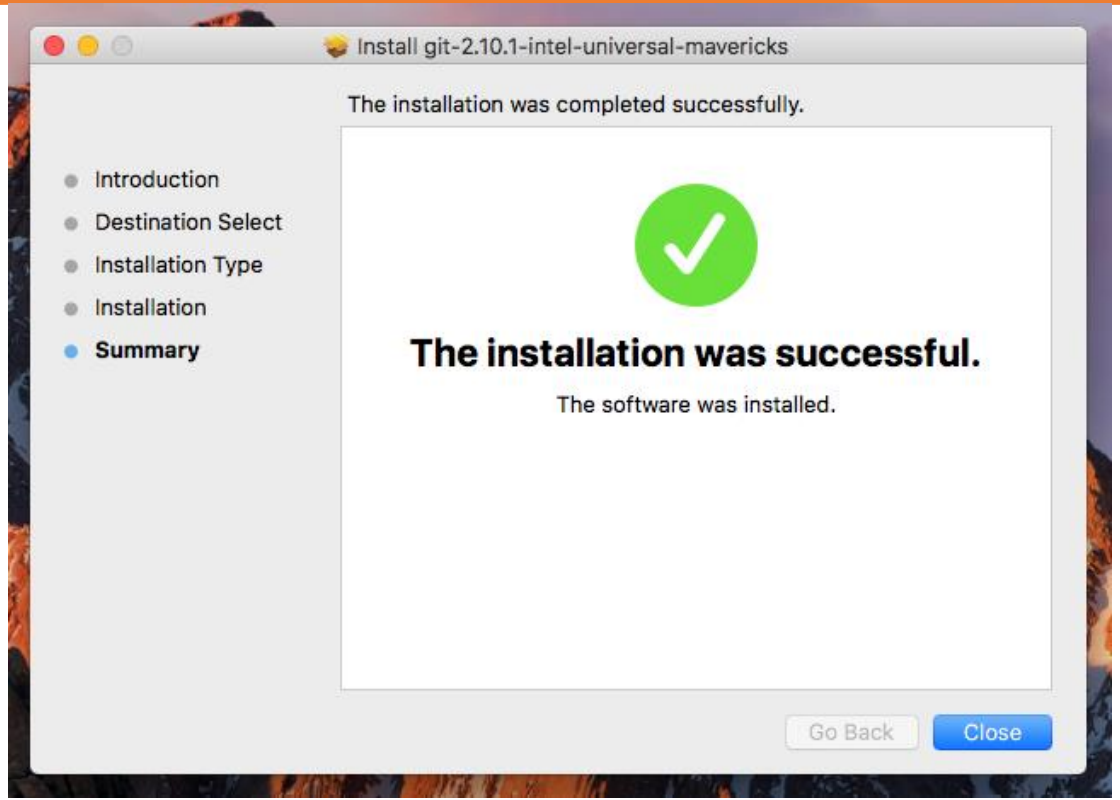
Installing Git on macOS



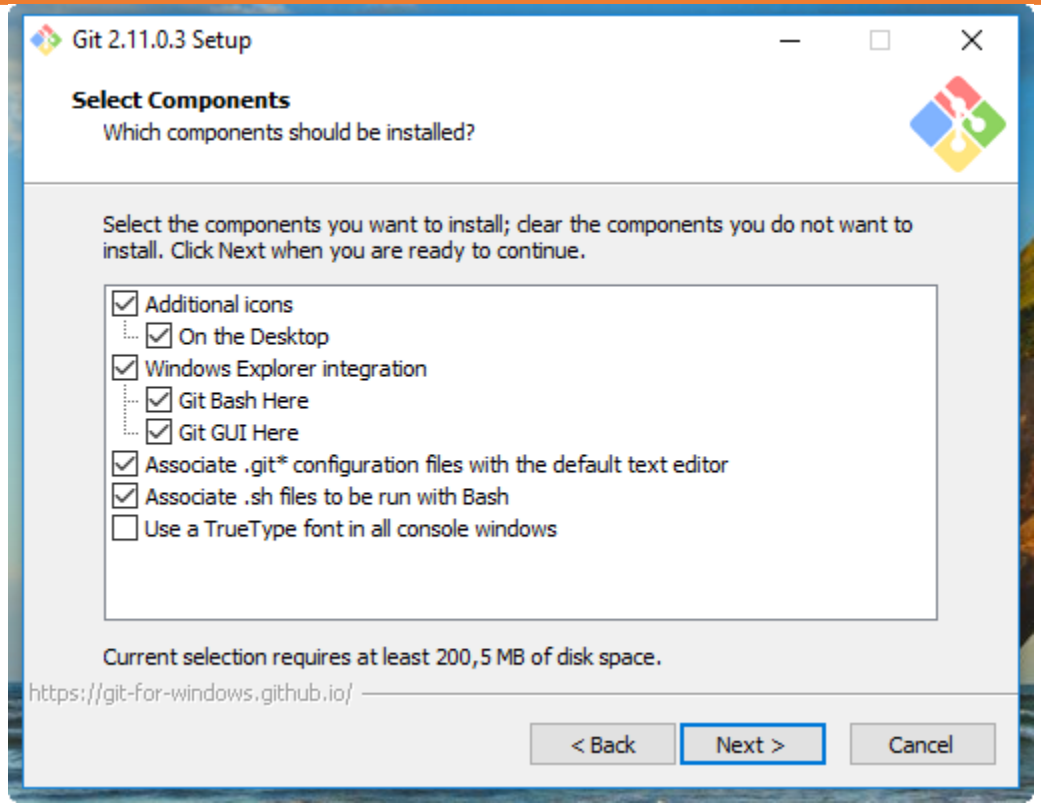
Here you can change the installation location, if you need; if in doubt, simply click Install

Installing Git on macOS

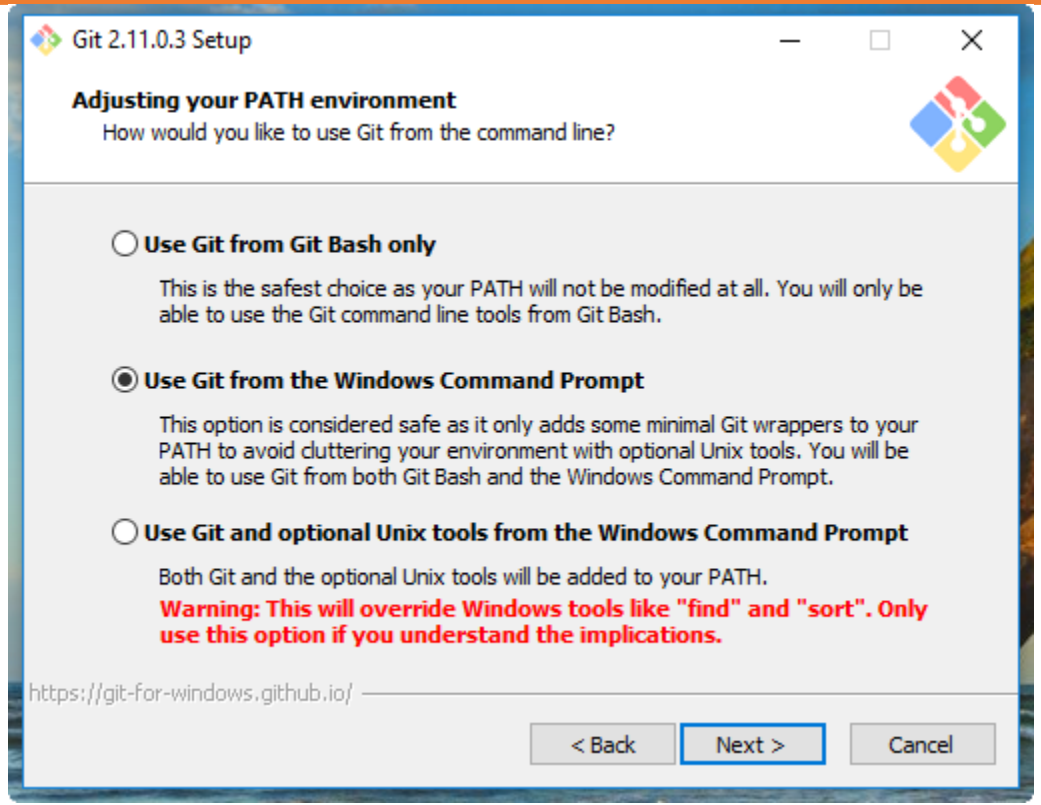
Installation complete



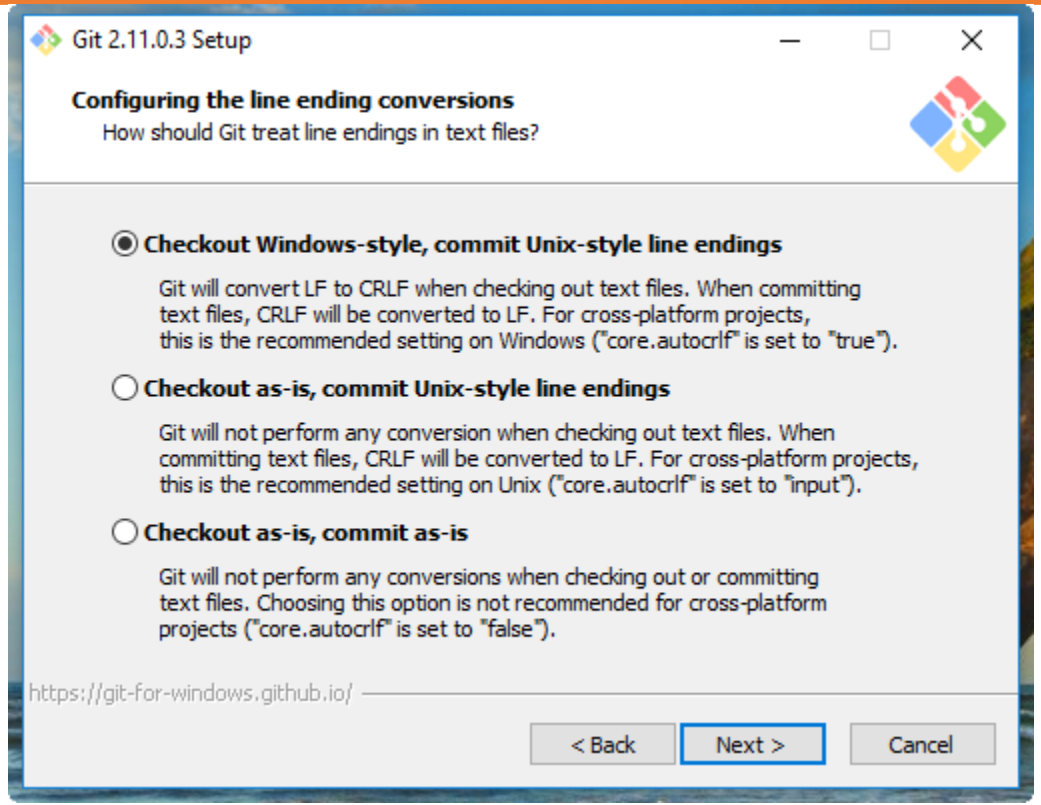
Installing Git on Windows



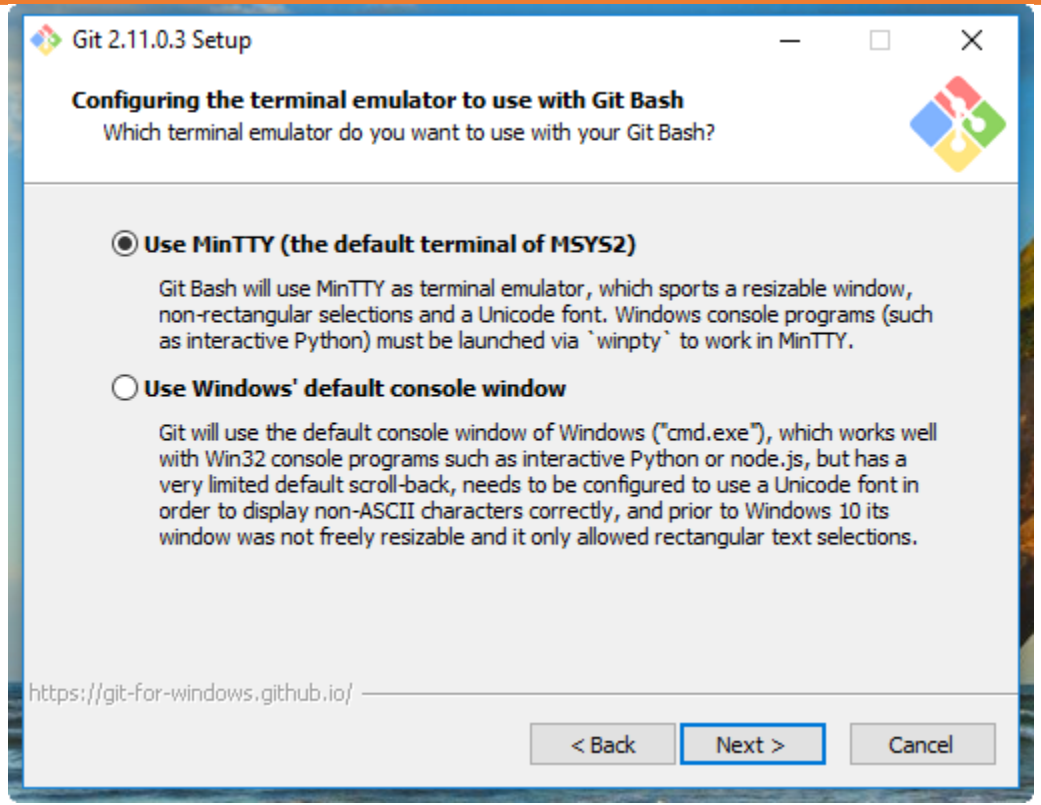
Installing Git on Windows



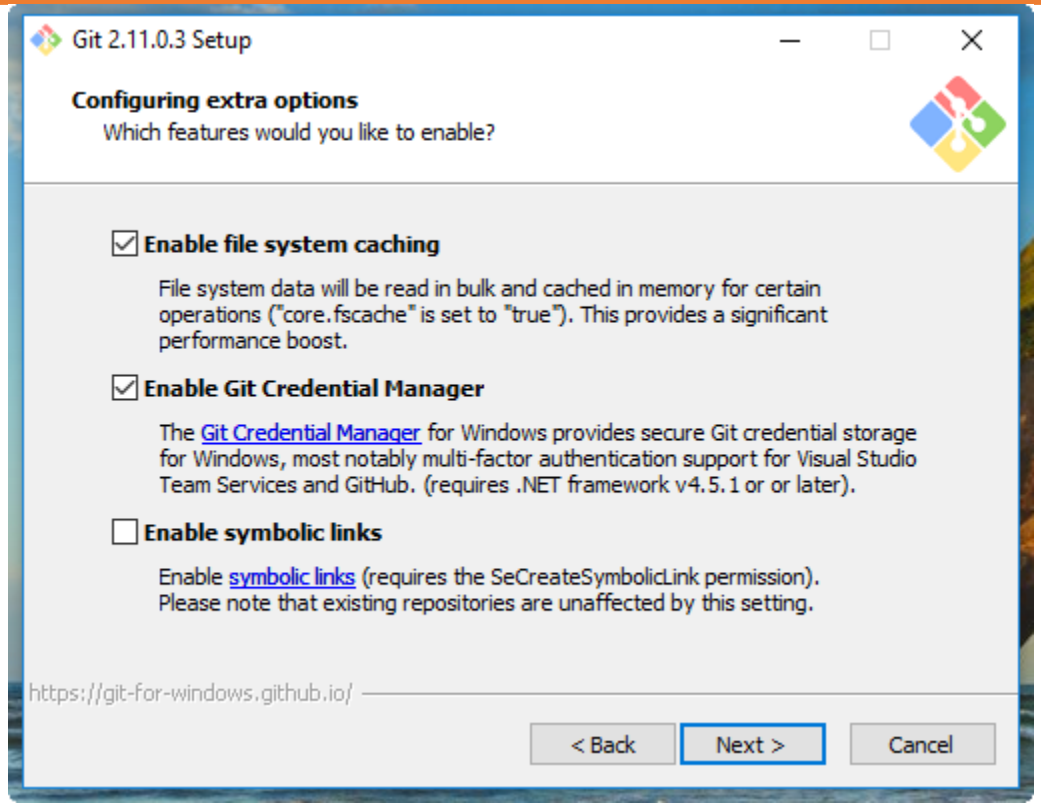
Installing Git on Windows



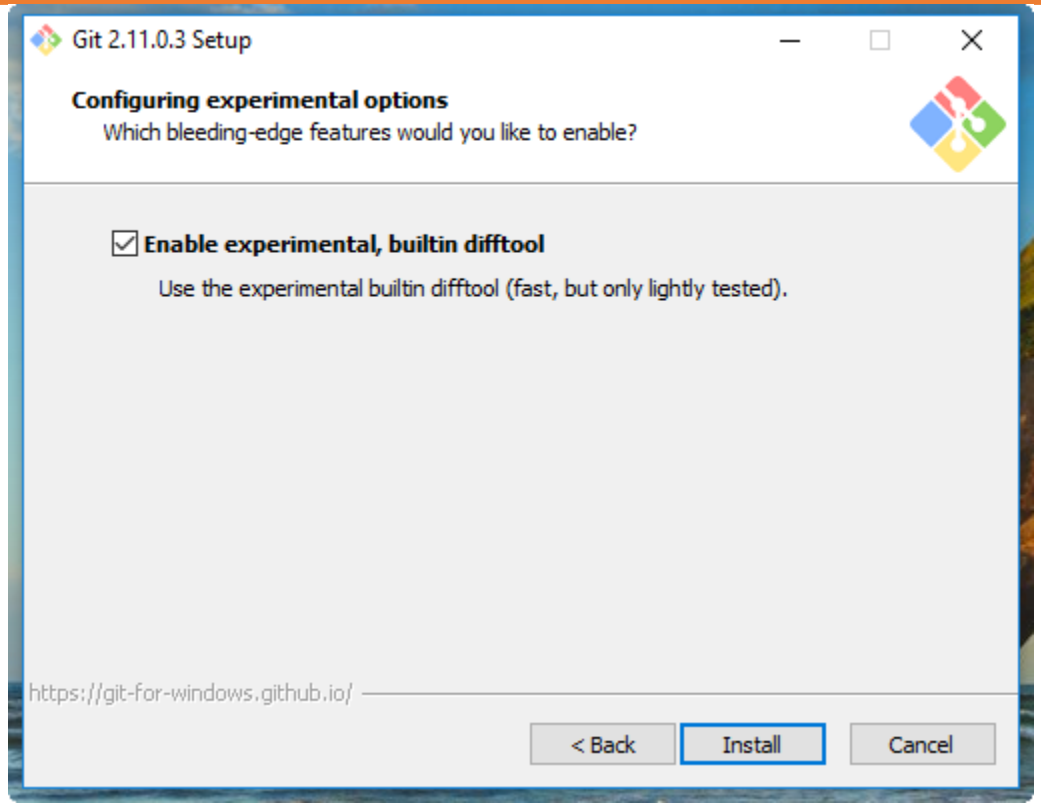
Installing Git on Windows



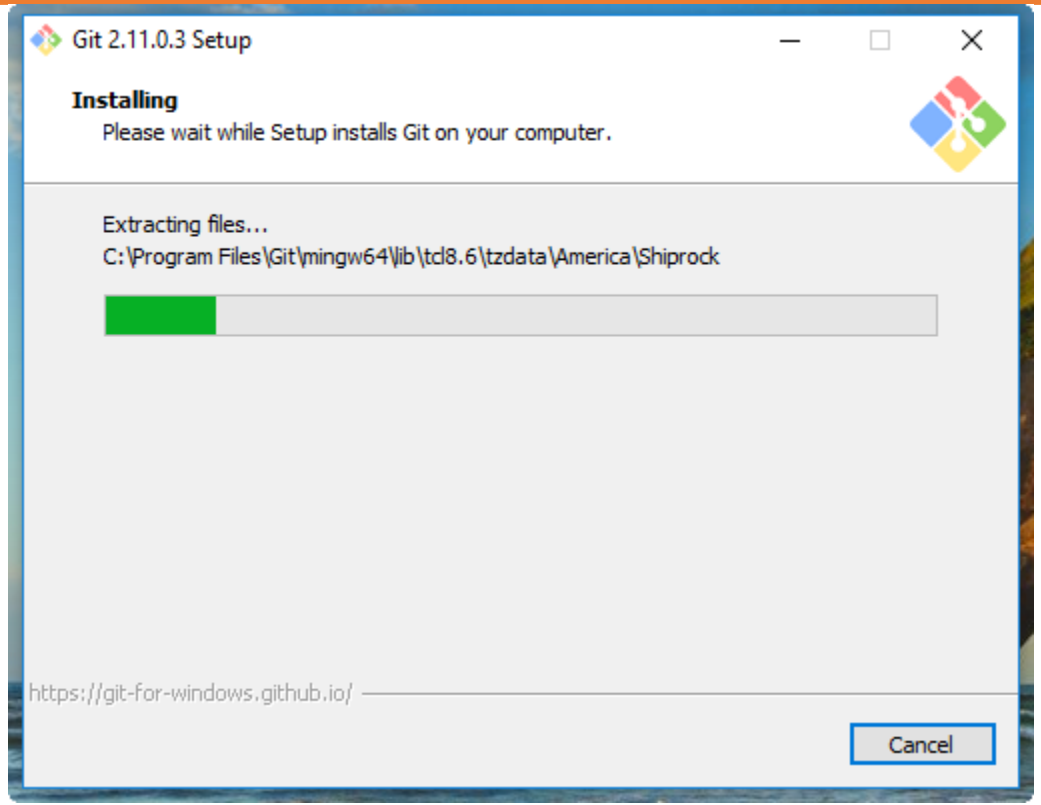
Installing Git on Windows



Installing Git on Windows

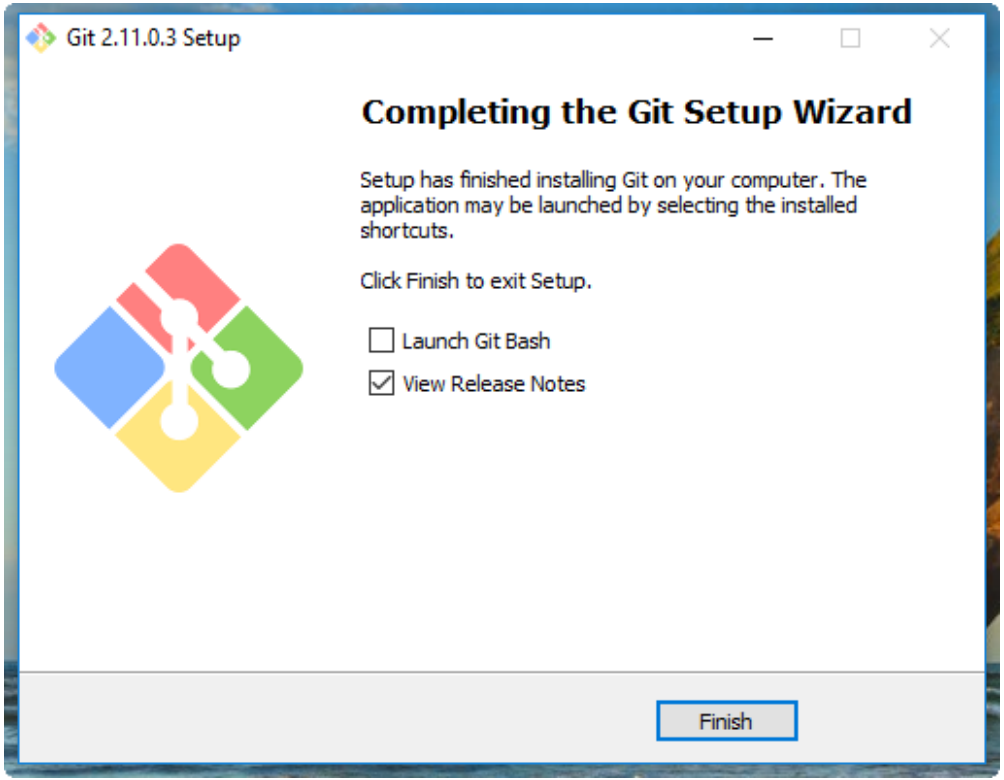


Installing Git on Windows



Installing Git on Windows

Git for Windows will install it in the default Program Files folder, as all the Windows programs usually do.



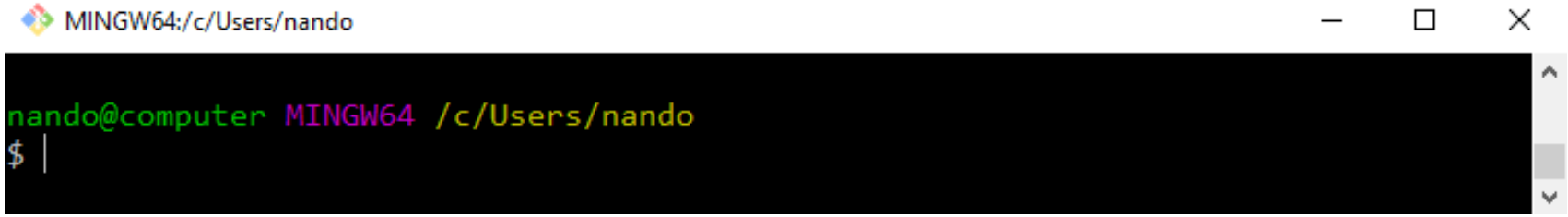
Running our first Git command

- Regardless of the OS in our screenshots, the git commands will work on Mach, Windows, Linux, etc.
- It's time to test our installation.
- Is Git ready to rock?
- ***Let's find out!***

Running our first Git command

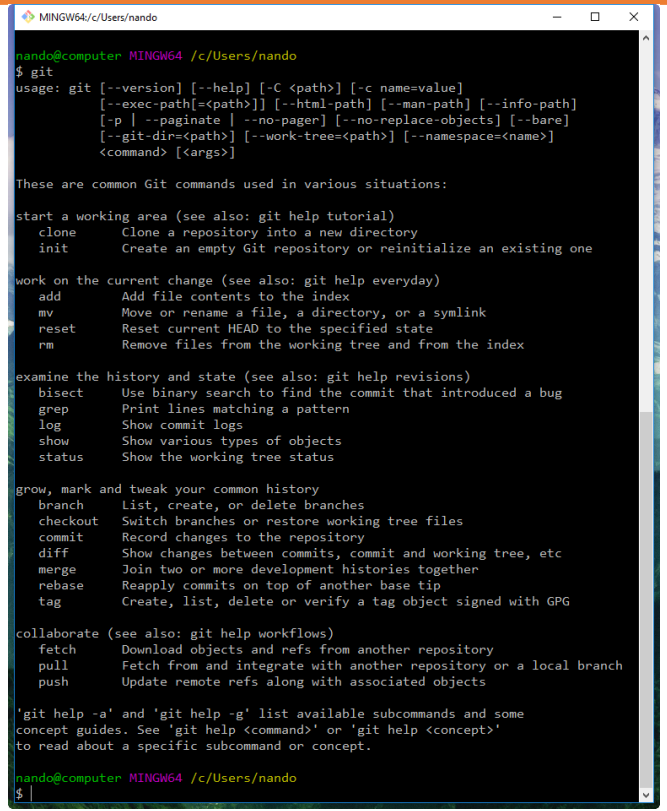
- Using shell integration, right-click on an empty place on the desktop and choose the new menu item Git Bash Here.

- It will appear as a new MinTTY shell, providing you a Git-ready bash for Windows:



Running our first Git command

- Now that we have a new, shiny Bash prompt, simply type **git** (or the equivalent, **git --help**



Making presentations

- Git needs to know who you are this is because in Git, every modification you make in a repository has to be signed with the name and email of the author so, before doing anything else, we have to tell Git this information.
- Type these two commands:

MINGW64:/home

```
nando@computer MINGW64 ~
$ git config --global user.name "Ferdinando Santacroce"

nando@computer MINGW64 ~
$ git config --global user.email "ferdinando.santacroce@gmail.com"

nando@computer MINGW64 ~
$
```

Setting up a new repository

- The first step is to set up a new repository.
- A repository is a container for your entire project; every file or subfolder within it belongs to that repository, in a consistent manner.
- Physically, a repository is nothing other than a folder that contains a special .git folder, the folder where the magic happens.

Setting up a new repository

MINGW64:/c/Repos/MyFirstRepo

```
[1] ~
$ cd /c/Repos/MyFirstRepo/

[2] /c/Repos/MyFirstRepo
$ git init
Initialized empty Git repository in C:/Repos/MyFirstRepo/.git/

[3] /c/Repos/MyFirstRepo (master)
$
```

Setting up a new repository

This PC ▸ Local Disk (C:) ▸ Repos ▸ MyFirstRepo ▸ .git

- Favorites
  - Desktop
  - Downloads
  - Recent places
  - Dropbox
  - Box Sync
- This PC

Name	Date modified	Type	Size
hooks	15/10/2014 23:05	File folder	
info	15/10/2014 23:05	File folder	
objects	15/10/2014 23:05	File folder	
refs	15/10/2014 23:05	File folder	
config	15/10/2014 23:05	File	1 KB
description	15/10/2014 23:05	File	1 KB
HEAD	15/10/2014 23:05	File	1 KB

Adding a file

- Let's create a text file, just to give it a try:

MINGW64:/c/Repos/MyFirstRepo

```
[3] /c/Repos/MyFirstRepo (master)
$ echo "Git rocks!" >> file.txt

[4] /c/Repos/MyFirstRepo (master)
$
```

Adding a file

- Okay, back to the topic. I want file.txt under the control of Git, so let's add it, as shown here:

```
MINGW64:/c/Repos/MyFirstRepo
[4] /c/Repos/MyFirstRepo (master)
$ git add file.txt

[5] /c/Repos/MyFirstRepo (master)
$
```

Adding a file

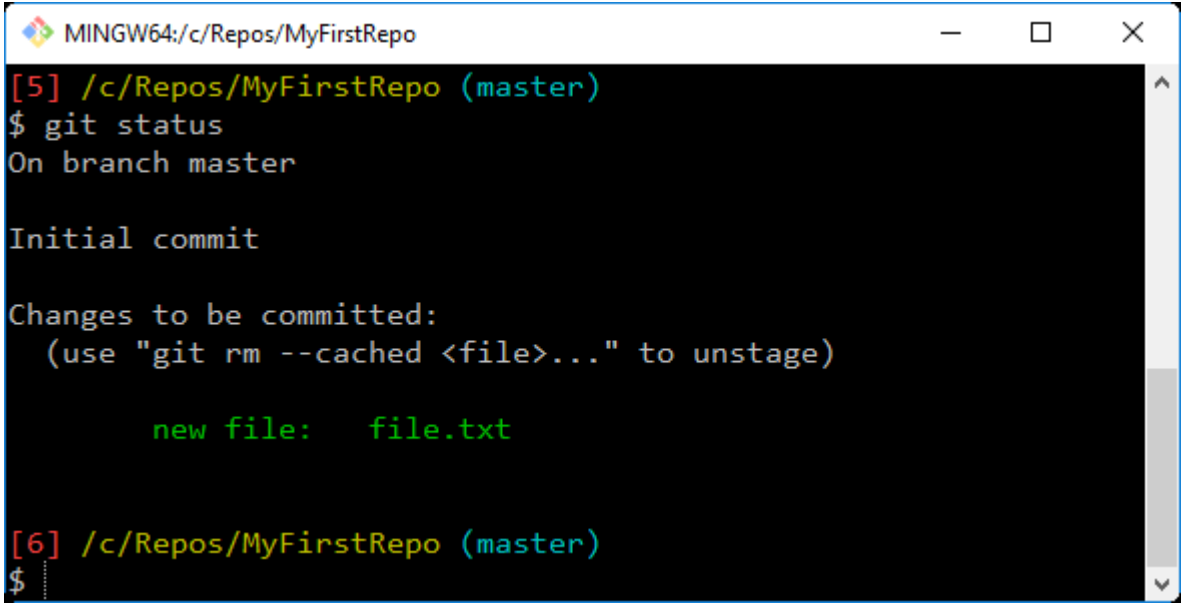
- In response to this command, it could happen that you will see this response message from Git:

warning: LF will be replaced by CRLF in file.txt.

- The file will have its original line endings in your working directory.

Adding a file

- Using the git status command, we can check the status of the repository, as shown in this screenshot:



Committing the added file

- At this point, Git knows about file.txt, but we have to perform another step to fix the snapshot of its content.
- We have to commit it using the appropriate git commit command.
- This time, we will add some flavor to our command, using the --message (or -m) subcommand, as shown here:

MINGW64:/c/Repos/MyFirstRepo

```
[6] /c/Repos/MyFirstRepo (master)
$ git commit --message "First commit"
[master (root-commit) 356a954] First commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt

[7] /c/Repos/MyFirstRepo (master)
$
```

Modifying a committed file

- Now, we can try to make some modifications to the file and see how to deal with it, as shown in the following screenshot:

MINGW64:/c/Repos/MyFirstRepo

```
[7] /c/Repos/MyFirstRepo (master)
$ echo "That's easy!" >> file.txt

[8] /c/Repos/MyFirstRepo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")

[9] /c/Repos/MyFirstRepo (master)
$
```

Modifying a committed file

- So, let's add the file again for the purpose of getting things ready for the next commit:

MINGW64:/c/Repos/MyFirstRepo

```
[9] /c/Repos/MyFirstRepo (master)
$ git commit
On branch master
Changes not staged for commit:
        modified:   file.txt

no changes added to commit

[10] /c/Repos/MyFirstRepo (master)
$ git add file.txt

[11] /c/Repos/MyFirstRepo (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   file.txt


[12] /c/Repos/MyFirstRepo (master)
$
```

Modifying a committed file

- Okay, let's make another commit, this time, avoiding the --message subcommand.

- Type git commit and hit the Enter key:

Modifying a committed file

- Fasten your seatbelts! You are now entering in a piece of code history!

MINGW64:/c/Repos/MyFirstRepo

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   file.txt
#
~
~
~
/c/Repos/MyFirstRepo/.git/COMMIT_EDITMSG [unix] (16:56 21/01/2017)1,0-1 All
```

Modifying a committed file

MINGW64:/c/Repos/MyFirstRepo

```
Thing are becoming serious :D
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   file.txt
#
~
~
~
<Repos/MyFirstRepo/.git/COMMIT_EDITMSG[+] [unix] (16:56 21/01/2017)1,30 All
-- INSERT --
```

Modifying a committed file

- You can also type the command in pairs as :wq, as we do in this screenshot, or use the equivalent :x command:

```
MINGW64:/c/Repos/MyFirstRepo
Thing are becoming serious :D
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   file.txt
#
~
~
~
<Repos/MyFirstRepo/.git/COMMIT_EDITMSG[+] [unix] (16:56 21/01/2017)1,29 All
:wq
```

Modifying a committed file

- After that, press Enter and another commit is done, as shown here:

MINGW64:/c/Repos/MyFirstRepo

```
[11] /c/Repos/MyFirstRepo (master)
$ git commit
[master 1a0bf2e] Thing are becoming serious :D
 1 file changed, 1 insertion(+)

[13] /c/Repos/MyFirstRepo (master)
$
```

Summary

- In this lesson, you learned that Git is not so difficult to install, even on a non-Unix platform, such as Windows.
- Once you have chosen a directory to include in a Git repository, you can see that initializing a new Git repository is as easy as executing a git init command.

2. Git Fundamentals - Working Locally

Git Fundamentals - Working Locally

- In this lesson, we will dive deep into some of the fundamentals of Git; it is essential to understand well how Git thinks about files, its way of tracking the history of commits, and all the basic commands that we need to master, in order to become proficient.

Digging into Git internals

- I want to show you how Git works internally with only the help of the shell, allowing you to follow all the steps on your computer and hoping that these will be clear enough for you to understand.

Git objects

- Let's create a new repository to refresh our memory and then start learning a little bit more about Git.
- In this example, we use Git to track our shopping list before going to the grocery; so, create a new grocery folder, and then initialize a new Git repository:

```
[1] ~$ mkdir grocery
[2] ~$ cd grocery/
[3] ~/grocery$ git init
Initialized empty Git repository in C:/Users/san/Google
Drive/LV/PortableGit/home/grocery/.git/
```

Git objects

- As we have already seen before, the result of the git init command is the creation of a .git folder, where Git stores all the files it needs to manage our repository:

[4] ~/grocery (master)$ ***ls -althr***
total 8drwxr-xr-x 1 san 1049089 0 Aug 17 11:11 ./drwxr-xr-x 1 san 1049089 0 Aug 17 11:11 ../drwxr-xr-x 1 san 1049089 0 Aug 17 11:11 .git/

Git objects

- Go on and create a new README.md file to remember the purpose of this repository:

```
[5] ~/grocery (master)$ echo "My shopping list repository"
> README.md
```

- Then add a banana to the shopping list:

```
[6] ~/grocery (master)$ echo "banana" > shoppingList.txt
```

Git objects

- At this point, as you already know, before doing a commit, we have to add files to the staging area; add both the files using the shortcut git add :

[7] ~/grocery (master)$ git add .

- With this trick (the dot after the git add command), you can add all the new or modified files in one shot.
- At this point, if you didn't set up a global username and email like we did in lesson1, Getting Started with Git, this is a thing that could happen:

Refer to the file 2_1.txt: https://jmp.sh/2g5H62t

Git objects

- So, let's change these settings and amend our commit (amending a commit is a way to redo the last commit and fix up some little mistakes, such as adding a forgotten file, changing the message or the author, as we are going to do; later we will learn in detail what this means):

[9] ~/grocery (master)$ git config user.name “Ernesto Lee“

[10] ~/grocery (master)$ git config user.email socrates73@gmail.com

Git objects

- For the purpose of this exercise, please leave the message as it is, press Esc, and then input the :wq (or :x) command and press Enter to save and exit:

[11] ~/grocery (master)$ git commit --amend --reset-author
#here Vim opens[master a57d783] Add a banana to the shopping list 2 files changed, 2 insertions(+) create mode 100644 README.md create mode 100644 shoppingList.txt

Git objects

- Now it's time to start investigating commits.
- To verify the commit we have just created, we can use the git log command:

```
[12] ~/grocery (master)$ git log
commit a57d783905e6a35032d9b0583f052fb42d5a1308
Author: Ernesto Lee socrates73@gmail.com
Date:   Thu Aug 17 13:51:33 2017 +0200Add a banana to the shopping
list
```

Git objects

- Just under the author and date, after a blank line, we can see the message we attached to the commit we made; even the message is part of the commit itself but there's something more under the hood; let's try to use the git log command with the --format=fuller option:

```
[13] ~/grocery (master)$ git log --format=fuller
commit a57d783905e6a35032d9b0583f052fb42d5a1308
Author: Ernesto Lee <socrates73@gmail.com>AuthorDate: Thu Aug 17
13:51:33 2017 +0200Commit: Ernesto Lee socrates73@gmail.com
CommitDate: Thu Aug 17 13:51:33 2017 +0200Add a banana to the
shopping list
```

Git objects

- We analyzed a commit, and the information supplied by a simple git log; but we are not yet satisfied, so go deeper and see what's inside.
- Using the git log command again, we can enable x-ray vision using the --format=raw option:

```
[14] ~/grocery (master)$ git log --format=raw
commit a57d783905e6a35032d9b0583f052fb42d5a1308tree
a31c31cb8d7cc16eeae1d2c15e61ed7382cebf40
author Ernesto Lee <socrates73@gmail.com> 1502970693
+0200committer Ernesto Lee <socrates73@gmail.com> 1502970693
+0200Add a banana to the shopping list
```

Git objects

- Back on topic; type the command, specifying the first characters of the commit's hash (a57d7 in my case):

```
[15] ~/grocery (master)$ git cat-file -p a57d7
tree a31c31cb8d7cc16eeae1d2c15e61ed7382cebf40author
Ernesto Lee <socrates73@gmail.com> 1502970693
+0200committer Ernesto Lee <socrates73@gmail.com>
1502970693 +0200Add a banana to the shopping list
```

Git objects

Porcelain commands and plumbing commands

- Git, as we know, has a myriad of commands, some of which are practically never used by the average user; as by example, the previous git cat-file.

- These commands are called plumbing commands, while those we have already learned about, such as git add, git commit, and so on, are among the so-called porcelain commands.

Git objects

- Here, for convenience, there is the output of the git cat-file -p command typed before:

```
[15] ~/grocery (master)$ git cat-file -p a57d7
tree
a31c31cb8d7cc16eeae1d2c15e61ed7382cebf40author
Ernesto Lee <socrates73@gmail.com> 1502970693
+0200committer Ernesto Lee <socrates73@gmail.com>
1502970693 +0200
```

Trees

- The tree is a container for blobs and other trees.
- The easiest way to understand how it works is to think about folders in your operating system, which also collect files and other subfolders inside them.
- Let's try to see what this additional Git object holds, using again the git cat-file -p command:

- [16] ~/grocery (master)$ git cat-file -p a31c3100644 blob907b75b54b7c70713a79cc6b7b172fb131d3027d README.md100644 blob 637a09b86af61897fb72f26bfb874f2ae726db82 shoppingList.txt

Blobs

```
[17] ~/grocery (master)$ git cat-file -p 637a0
banana
```

- Wow! Its content is exactly the content of our shoppingFile.txt file.
- To confirm, we can use the cat command, which on *nix systems allows you to see the contents of a file:

```
[18] ~/grocery (master)$ cat shoppingList.txt
banana
```

Blobs

- Blobs are binary files, nothing more and nothing less.
- These byte sequences, which cannot be interpreted with the naked eye, retain inside information belonging to any file, whether binary or textual, images, source code, archives, and so on.
- Everything is compressed and transformed into a blob before archiving it into a Git repository.

Blobs

- Let's try to understand it better with an example.
- Open a shell and try to play a bit with another plumbing command, git hash-object:

```
[19] ~/grocery (master)$ echo "banana" | git hash-object –stdin
637a09b86af61897fb72f26bfb874f2ae726db82
```

Even deeper - the Git storage object model

- Do you remember the .git folder? Let's put our nose inside it:

```
[20] ~/grocery (master)$ ls -althr
```

Even deeper - the Git storage object model

- Within it, there is an objects subfolder; let's take a look:

[21] ~/grocery (master) $ ll
.***git/objects/*** total 4drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 ./drwxr-xr-x 1 san 1049089 0 Aug 18 17:22 ../drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 63/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 90/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 a3/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 a5/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 c7/drwxr-xr-x 1 san 1049089 0 Aug 17 11:11 info/drwxr-xr-x 1 san 1049089 0 Aug 18 17:12 pack/

Even deeper - the Git storage object model

- Other than info and pack folders, which are not interesting for us right now, as you can see there are some other folders with a strange two-character name; let's go inside the 63 folder:

```
[22] ~/grocery (master)
$ ll .git/objects/63/
total 1
drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 ./
drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 ../
-r--r--r-- 1 san 1049089 20 Aug 17 13:34
7a09b86af61897fb72f26bfb874f2ae726db82
```

Even deeper - the Git storage object model

- To become aware of this, we need a new commit. So, let's now proceed modifying the shoppingList.txt file:

```
[23] ~/grocery (master)$ echo "apple" >>
shoppingList.txt
[24] ~/grocery (master)$ git add shoppingList.txt
[25] ~/grocery (master)$ git commit -m "Add an apple“
[master e4a5e7b] Add an apple 1 file changed, 1
insertion(+)
```

Even deeper - the Git storage object model

- Use the git log command to check the new commit; the --oneline option allows us to see the log in a more compact way:

```
[26] ~/grocery (master)$ git log –oneline
e4a5e7b Add an applea57d783 Add a banana to the shopping list
```

Even deeper - the Git storage object model

- Okay, we have a new commit, with its hash.
- Time to see what's inside it:

```
[27] ~/grocery (master)$ git cat-file -p e4a5e7b
tree 4c931e9fd8ca4581ddd5de9efd45daf0e5c300a0parent
a57d783905e6a35032d9b0583f052fb42d5a1308author
Ernesto Lee <socrates73@gmail.com> 1503586854
+0200committer Ernesto Lee <socrates73@gmail.com>
1503586854 +0200Add an apple
```

Git doesn't use deltas

- Instead, in Git even if you change only a char in a big text file, it always stores a new version of the file: Git doesn't do deltas (at least not in this case), and every commit is actually a snapshot of the entire repository.
- At this point, people usually exclaim:

"Gosh, Git waste a large amount of disk space in vain!"

- Well, this is simply untrue.

Git doesn't use deltas

- Furthermore, Git has a clever way to deal with files; let's take a look again at the last commit:

```
[28] ~/grocery (master)$ git cat-file -p e4a5e7b
tree 4c931e9fd8ca4581ddd5de9efd45daf0e5c300a0parent
a57d783905e6a35032d9b0583f052fb42d5a1308author
Ernesto Lee <socrates73@gmail.com> 1503586854
+0200committer Ernesto Lee <socrates73@gmail.com>
1503586854 +0200Add an apple
```

Git doesn't use deltas

- Okay, now to the tree:

```
[29] ~/grocery (master)$ git cat-file -p 4c931e9100644
blob 907b75b54b7c70713a79cc6b7b172fb131d3027d
README.md100644 blob
e4ceb844d94edba245ba12246d3eb6d9d3aba504
shoppingList.txt
```

Git doesn't use deltas

- Annotate the two hashes on a notepad; now we have to look at the tree of the first commit; cat-file the commit:

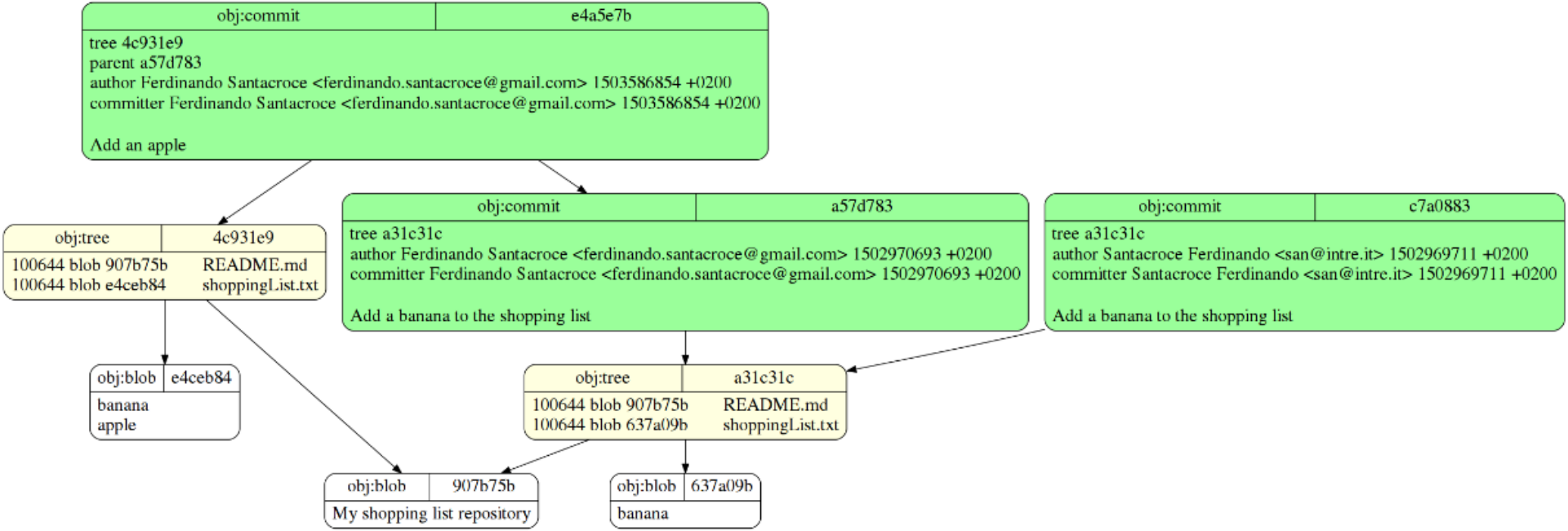
```
[30] ~/grocery (master)$ git cat-file -p a57d783
tree a31c31cb8d7cc16eeae1d2c15e61ed7382cebf40author
Ernesto Lee <socrates73@gmail.com> 1502970693
+0200committer Ernesto Lee <socrates73@gmail.com>
1502970693 +0200
Add a banana to the shopping list
```

Git doesn't use deltas

- Then cat-file the tree:

```
[31] ~/grocery (master)$ git cat-file -p a31c31c100644
blob 907b75b54b7c70713a79cc6b7b172fb131d3027d
README.md100644 blob
637a09b86af61897fb72f26bfb874f2ae726db82
shoppingList.txt
```

Wrapping up


obj:commit
e4a5e7b
tree 4c931e9
parent a57d783
author Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1503586854 +0200
committer Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1503586854 +0200
Add an apple
obj:tree
4c931e9
100644 blob 907b75b README.md
100644 blob e4ceb84 shoppingList.txt
obj:commit
a57d783
tree a31c31c
author Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1502970693 +0200
committer Ferdinando Santacroce <ferdinando.santacroce@gmail.com> 1502970693 +0200
Add a banana to the shopping list
obj:commit
c7a0883
tree a31c31c
author Santacroce Ferdinando <san@intre.it> 1502969711 +0200
committer Santacroce Ferdinando <san@intre.it> 1502969711 +0200
Add a banana to the shopping list
obj:blob
e4ceb84
banana
apple
obj:tree
a31c31c
100644 blob 907b75b README.md
100644 blob 637a09b shoppingList.txt
obj:blob
907b75b
My shopping list repository
obj:blob
637a09b
banana

Wrapping up

- In this graphic representation of previous slide, you will find a detailed diagram that represents the current structure of the newly created repository; you can see trees (yellow), blobs (white), commits (green), and all relationships between them, represented by oriented arrows.

Git references

It's all about labels

- In Git, a branch is nothing more than a label, a mobile label placed on a commit.

- In fact, every leaf on a Git branch has to be labeled with a meaningful name to allow us to reach it and then move around, go back, merge, rebase, or discard some commits when needed.

Git references

- Let's start exploring this topic by checking the current status of our grocery repository; we do it using the well-known git log command, this time adding some new options:

[1] ~/grocery (master)$ git log --oneline --graph --decorate
e4a5e7b
(HEAD -> master) Add an apple* a57d783 Add a banana
to the shopping list

Git references

- We'll now do a new commit and see what happens:

```
[2] ~/grocery (master)$ echo "orange" >>
shoppingList.txt
[3] ~/grocery (master)$ git commit -am "Add an orange“
[master 0e8b5cf] Add an orange 1 file changed, 1
insertion(+)
```

Git references

- Okay, go on now and take a look at the current repository situation:

[4] ~/grocery (master)$ git log --oneline --graph –decorate –all
* 0e8b5cf (HEAD -> master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- But how will Git handle this feature? Let's go back to putting the nose again in the .git folder:

[5] ~/grocery (master)$ ll .git/

total 21drwxr-xr-x 1 san 1049089 0 Aug 25 11:20 ./drwxr-xr-x 1 san 1049089 0 Aug 25 11:19 ../-rw-r--r-- 1 san 1049089 14 Aug 25 11:20 COMMIT_EDITMSG-rw-r--r-- 1 san 1049089 208 Aug 17 13:51 config-rw-r--r-- 1 san 1049089 73 Aug 17 11:11 description-rw-r--r-- 1 san 1049089 23 Aug 17 11:11 HEADdrwxr-xr-x 1 san 1049089 0 Aug 18 17:15 hooks/-rw-r--r-- 1 san 1049089 217 Aug 25 11:20 indexdrwxr-xr-x 1 san 1049089 0 Aug 18 17:15 info/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 logs/drwxr-xr-x 1 san 1049089 0 Aug 25 11:20 objects/drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 refs/

Git references

- There's a refs folder: let's take a look inside:

```
[6] ~/grocery (master)$ ll .git/refs/
total 4drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 ./drwxr-xr-x 1 san 1049089 0 Aug 25 11:20 ../drwxr-xr-x 1 san 1049089 0 Aug 25 11:20 heads/drwxr-xr-x 1 san 1049089 0 Aug 17 11:11 tags/
```

Git references

- Now go to heads:

[7] ~/grocery (master)$ ll .git/refs/heads/
total 1drwxr-xr-x 1 san 1049089 0 Aug 25 11:20 ./drwxr-xr-x 1 san 1049089 0 Aug 18 17:15 ../-rw-r--r-- 1 san 1049089 41 Aug 25 11:20 master

- There's a master file inside! Let's see what's the content:

[8] ~/grocery (master)$ cat .git/refs/heads/master
0e8b5cf1c1b44110dd36dea5ce0ae29ce22ad4b8

Git references

Creating a new branch

- Now that we have warmed up, the fun begins.
- Let's see what happens when you ask Git to create a new branch.
- Since we are going to serve a delicious fruit salad, it's time to set a branch apart for a berries-flavored variant recipe:

```
[9] ~/grocery (master)$ git branch berries
```

Git references

- So, git log again:

[10] ~/grocery (master)$ git log --oneline --graph –decorate –all
* 0e8b5cf (HEAD -> master, berries) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Anyway, at the moment we continue to be located in the master branch; in fact, as you can see in the shell output prompt, it continues to appear (master) between the round parenthesis:

[10] ~/grocery (master)

- How can I switch branch? By using the git checkout command:

[11] ~/grocery (master)$ git checkout berries
Switched to branch 'berries'

Git references

- Do a git log to see:

[12] ~/grocery (berries)$ git log --oneline --graph –decorate –all
* 0e8b5cf (HEAD -> berries, master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

HEAD, or you are here

- During previous exercises we continued to see that HEAD thing while using git log, and now it's time to investigate a little bit.
- First of all, what is HEAD? As branches are, HEAD is a reference. It represents a pointer to the place on where we are right now, nothing more, nothing less. In practice instead, it is just another plain text file:

```
[13] ~/grocery (berries)$ cat .git/HEAD
ref: refs/heads/berries
```

Git references

- Add a blackberry to the shopping list:

```
[14] ~/grocery (berries)$ echo "blackberry" >>
shoppingList.txt
```

- Then perform a commit:

```
[15] ~/grocery (berries)$ git commit -am "Add a
blackberry“
[berries ef6c382] Add a blackberry 1 file changed, 1
insertion(+)
```

Git references

- Take a look on what happened with the usual git log command:

[16] ~/grocery (berries)$ git log --oneline --graph –decorate –all

* ef6c382 (HEAD -> berries) Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

- Okay, so now our shoppingList.txt file appears to contain these text lines:

[17] ~/grocery (berries)$ cat shoppingList.txt

bananaappleorangeblackberry

Git references

- Check out the master branch:

[18] ~/grocery (berries)$ git checkout master
Switched to branch 'master’

- Look at the shoppingFile.txt content:

[19] ~/grocery (master)$ cat shoppingList.txt
bananaappleorange

Git references

- We actually moved back to where we were before adding the blackberry; as it is being added in the berries branch, here in the master branch it does not exist: sounds good, doesn't it?

- Even the HEAD file has been updated accordingly:

```
[20] ~/grocery (master)$ cat .git/HEAD
ref: refs/heads/master
```

Git references

- Go back to the repository now; let's do the usual git log:

[21] ~/grocery (master)$ git log --oneline --graph –decorate
* 0e8b5cf (HEAD -> master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Uh-oh: where is the berries branch? Don't worry: git log usually displays only the branch you are on, and the commit that belongs to it.
- To see all branches, you only need to add the --all option:

[22] ~/grocery (master)$ git log --oneline --graph --decorate –all

* ef6c382 (berries) Add a blackberry* 0e8b5cf (HEAD -> master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- First, check out the berries branch:

```
[23] ~/grocery (master)$ git checkout
-Switched to branch 'berries'
```

- Now a new command, git reset (please don't care about the --hard option for now):

```
[24] ~/grocery (berries)$ git reset --hard master
HEAD is now at 0e8b5cf Add an orange
```

Git references

- In Git, this is simple as this. The git reset actually moves a branch from the current position to a new one; here we said Git to move the current berries branch to where master is, and the result is that now we have all the two branches pointing to the same commit:

[25] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* 0e8b5cf (HEAD -> berries, master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- You can double-check this looking at refs files; this is the berries one:

```
[26] ~/grocery (berries)$ cat .git/refs/heads/berries
0e8b5cf1c1b44110dd36dea5ce0ae29ce22ad4b8
```

- And this is the master one:

```
[27] ~/grocery (berries)$ cat .git/refs/heads/master
0e8b5cf1c1b44110dd36dea5ce0ae29ce22ad4b8
```

Git references

- Let's try another trick: we can use git reset to move the actual branch directly to a commit.
- And to make things more interesting, let's try to point the blackberry commit (if you scroll your shell window backwards, you can see its hash, which for me is ef6c382) so, git reset to the ef6c382 commit:

```
[28] ~/grocery (berries)$ git reset --hard ef6c382
HEAD is now at ef6c382 Add a blackberry
```

Git references

- And then do the usual git log:

[29] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* ef6c382 (HEAD -> berries) Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Assume you want to add a watermelon to the shopping list, but later you realize you added it to the wrong berries branch; so, add "watermelon" to the shoppingList.txt file:

[30] ~/grocery (berries)$ echo "watermelon" >> shoppingList.txt

- Then do the commit:

[31] ~/grocery (berries)$ git commit -am "Add a watermelon“

[berries a8c6219] Add a watermelon 1 file changed, 1 insertion(+)

Git references

- And do a git log to check the result:

[32] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* a8c6219 (HEAD -> berries) Add a watermelon* ef6c382 Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Now our aim here is: have a new melons branch, which the watermelon commit have to belong to, then set the house in order and move the berries branch back to the blackberry commit.
- To keep the watermelon commit, first create a melon branch that points to it with the well-known git branch command:

```
[33] ~/grocery (berries)$ git branch melons
```

Git references

- Let's check:

[34] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* a8c6219 (HEAD -> berries, melons) Add a watermelon* ef6c382 Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- So, let's step back our berries branch using caret; do a git reset --hard HEAD^:

[35] ~/grocery (berries)$ git reset --hard HEAD^
HEAD is now at ef6c382 Add a blackberry

- Let's see the result:

[36] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* a8c6219 (melons) Add a watermelon* ef6c382 (HEAD -> berries) Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Just to remark concepts, let's take a look at the shoppingList.txt file here in the berries branch:

```
[37] ~/grocery (berries)$ cat shoppingList.txt
bananaappleorangeblackberry
```

- Okay, here we have blackberry, other than the other previously added fruits.
- Switch to master and check again; check out the master branch:

```
[38] ~/grocery (berries)$ git checkout master
Switched to branch 'master'
```

Git references

- Then cat the file:

```
[39] ~/grocery (master)$ cat shoppingList.txt
bananaappleorange
```

- Okay, no blackberry here, but only fruits added before the berries branch creation and now a last check on the melons branch; check out the branch:

```
[40] ~/grocery (master)$ git checkout melons
Switched to branch 'melons'
```

Git references

- And cat the shoppingList.txt file:

```
[41] ~/grocery (melons)$ cat shoppingList.txt
bananaappleorangeblackberrywatermelon
```

Git references

- For the sake of the explanation, go back to the master branch and see what happens when we check out the previous commit, moving HEAD backward; perform a git checkout HEAD^:

```
[42] ~/grocery (master)$ git checkout HEAD^
Note: checking out 'HEAD^'.
You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout. If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:  git checkout -b <new-branch-name>HEAD is now at e4a5e7b... Add an apple
```

Git references

- Here, Git says we are in a detached HEAD state. Being in this state basically means that HEAD does not reference a branch, but directly a commit, the e4a5e7b one in this case; do a git log and see:

[43] ~/grocery ((e4a5e7b...))$ git log --oneline --graph --decorate –all
* a8c6219 (melons) Add a watermelon* ef6c382 (berries) Add a blackberry* 0e8b5cf (master) Add an orange* e4a5e7b (HEAD) Ad an apple* a57d783 Add a banana to the shopping list

Git references

- First of all, in the shell prompt you can see that between rounds, that now are doubled, there is not a branch name, but the first seven characters of the commit, ((e4a5e7b...)).
- Then, HEAD is now stuck to that commit, while branches, especially the master one, remains at their own place.
- As a result, the HEAD file now contains the hash of that commit, not a ref to a branch as before:

```
[44] ~/grocery ((e4a5e7b...))$ cat .git/HEAD
e4a5e7b3c64bee8b60e23760626e2278aa322f05
```

Git references

- Okay, let's have some fun; modify the shoppingList.txt file, adding a bug:

[45] ~/grocery ((e4a5e7b...))$ echo "bug" > shoppingList.txt

- Then commit this voluntary mistake:

[46] ~/grocery ((e4a5e7b...))$ git commit -am "Bug eats all the fruits!“
[detached HEAD 07b1858] Bug eats all the fruits! 1 file changed, 1 insertion(+), 2 deletions(-)

Git references

- Let's cat the file:

[47] ~/grocery ((07b1858...))$ cat shoppingList.txt
bug
Ouch, we actually erased all your shopping list files!

- What happened in the repository then?

Refer to the file 2_2.txt

Git references

- Okay, so if we now check out master again, what happens? Give it a try:

[49] ~/grocery ((07b1858...))$ git checkout master
Warning: you are leaving 1 commit behind, not connected to any of your branches: 07b1858 Bug eats all the fruits! If you want to keep it by creating a new branch, this may be a good time to do so with:

 git branch <new-branch-name> 07b1858Switched to branch 'master'

Git references

- Let's check the situation:

[50] ~/grocery (master)$ git log --oneline --graph --decorate –all
* a8c6219 (melons) Add a watermelon* ef6c382 (berries) Add a blackberry* 0e8b5cf (HEAD -> master) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Let's try it, creating a bug branch:

[51] ~/grocery (master)$ git branch bug
07b1858

- Let's see what happened:

[52] ~/grocery (master)$ git log --oneline --graph --decorate –all
* 07b1858 (bug) Bug eats all the fruits!| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry| * 0e8b5cf (HEAD -> master) Add an orange|/* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- We can take a look at it with a convenient Git command, git reflog show:

[53] ~/grocery (master)$ git reflog show0e8b5cf HEAD@{0}: checkout: moving from 07b18581801f9c2c08c25cad3b43aeee7420ffdd to master07b1858 HEAD@{1}: commit: Bug eats all the fruits!e4a5e7b HEAD@{2}: checkout: moving from master to HEAD^0e8b5cf HEAD@{3}: reset: moving to 0e8b5cfe4a5e7b HEAD@{4}: reset: moving to HEAD^0e8b5cf HEAD@{5}: checkout: moving from melons to mastera8c6219 HEAD@{6}: checkout: moving from master to melons0e8b5cf HEAD@{7}: checkout: moving from berries to masteref6c382 HEAD@{8}: reset: moving to HEAD^a8c6219 HEAD@{9}: commit: Add a watermelonef6c382 HEAD@{10}: reset: moving to ef6c382ef6c382 HEAD@{11}: reset: moving to ef6c3820e8b5cf HEAD@{12}: reset: moving to masteref6c382 HEAD@{13}: checkout: moving from master to berries0e8b5cf HEAD@{14}: checkout: moving from berries to masteref6c382 HEAD@{15}: commit: Add a blackberry0e8b5cf HEAD@{16}: checkout: moving from master to berries0e8b5cf HEAD@{17}: commit: Add an orangee4a5e7b HEAD@{18}: commit: Add an applea57d783 HEAD@{19}: commit (amend): Add a banana to the shopping listc7a0883 HEAD@{20}: commit (initial): Add a banana to the shopping list

Git references

- Actually, here there are all the movements the HEAD reference made in my repository since the beginning, in reverse order, as you may have already noticed.

- In fact, the last one (HEAD@{0}) says:

checkout: moving from 07b18581801f9c2c08c25cad3b43aeee7420ffdd to master

Git references

```
[54] ~/grocery (master)
$ git reflog berries

ef6c382 berries@{0}: reset: moving to HEAD^a8c6219
berries@{1}: commit: Add a watermelonef6c382
berries@{2}: reset: moving to ef6c3820e8b5cf
berries@{3}: reset: moving to masteref6c382
berries@{4}: commit: Add a blackberry0e8b5cf
berries@{5}: branch: Created from master
```

Git references

- Creating a tag is simple: you only need the git tag command, followed by a tag name; we can create one in the tip commit of bug branch to give it a try; check out the bug branch:

```
[1] ~/grocery (master)$ git checkout bug
Switched to branch 'bug'
```

- Then use the git tag command followed by the funny bugTag name:

```
[2] ~/grocery (bug)$ git tag bugTag
```

Git references

- Let's see what git log says:

[3] ~/grocery (bug)$ git log --oneline --graph --decorate –all

* 07b1858 (HEAD -> bug, tag: bugTag) Bug eats all the fruits!| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry| * 0e8b5cf (master) Add an orange

* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- If you do a commit in this branch, you will see the bugTag will remain at its place; add a new line to the same old shopping list file:

```
[4] ~/grocery (bug)$ echo "another bug" >> shoppingList.txt
```

- Perform a commit:

```
[5] ~/grocery (bug)$ git commit -am "Another bug!“
[bug 5d605c6] Another bug! 1 file changed, 1 insertion(+)
```

Git references

- Then look at the current situation:

[6] ~/grocery (bug)$ git log --oneline --graph --decorate –all
* 5d605c6 (HEAD -> bug) Another bug!* 07b1858 (tag: bugTag) Bug eats all the fruits!| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry| * 0e8b5cf (master) Add an orange|/* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Git references

- Even tags are references, and they are stored, as branches, as simple text files in the tags subfolder within the .git folder; take a look under the .git/refs/tags folder, you will see a bugTag file; look at the content:

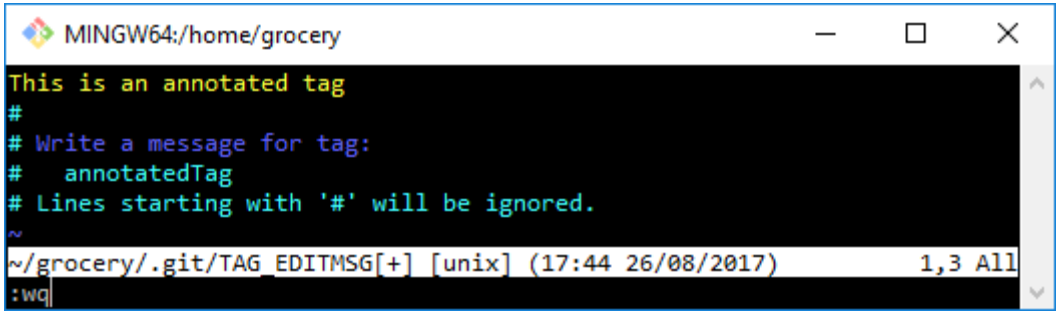
```
[7] ~/grocery (bug)$ cat .git/refs/tags/bugTag
07b18581801f9c2c08c25cad3b43aeee7420ffdd
```

Git references

- To create one, simply append -a to the command; let's create another one to give this a try:

[8] ~/grocery (bug)$ git tag -a annotatedTag 07b1858

- At this point Git opens the default editor, to allow you to write the tag message, as in the following screenshot:



Git references

- Save and exit, and then see the log:

```
[9] ~/grocery (bug)$ git log --oneline --graph --decorate –all
* 5d605c6 (HEAD -> bug) Another bug!* 07b1858 (tag:
bugTag, tag: annotatedTag) Bug eats all the fruits!| * a8c6219
(melons) Add a watermelon| * ef6c382 (berries) Add a
blackberry| * 0e8b5cf (master) Add an orange
|/
* e4a5e7b Add an apple* a57d783 Add a banana to the
shopping list
```

Git references

- A new ref has been created:

```
[10] ~/grocery (bug)$ cat .git/refs/tags/annotatedTag
17c289ddf23798de6eee8fe6c2e908cf0c3a6747
```

- But even a new object: try to cat-file the hash you see in the reference:

```
[11] ~/grocery (bug)$ git cat-file -p 17c289object
07b18581801f9c2c08c25cad3b43aeee7420ffddtype committag
annotatedTagtagger Ernesto Lee <socrates73@gmail.com>
150376226 4 +0200This is an annotated tag
```

Staging area, working tree, and HEAD commit

- Let's focus on this right now; move to the master branch, if not already there, then type the git status command; it allows us to see the actual status of the staging area:

```
[1] ~/grocery (master)$ git statusOn branch master
nothing to commit, working tree clean
```

Staging area, working tree, and HEAD commit

- Git says there's nothing to commit, our working tree is clean.

- But what's a working tree? Is it the same as the working directory we talked about? Well, yes and no, and it's confusing, I know.

- Git had (and still have) some troubles with names; in fact, as we said a couple of lines before, even for the staging area we have two names (the other one is index).

Staging area, working tree, and HEAD commit

- Add a peach to the shoppingList.txt file:

[2] ~/grocery (master)$ echo "peach" >> shoppingList.txt

- Then make use of this new learnt command again, git status:

[3] ~/grocery (master)$ git statusOn branch master Changes not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory)modified: shoppingList.txtno changes added to commit (use "git add" and/or "git commit -a")

Staging area, working tree, and HEAD commit

- Let's try to add it; we will see the second option later.
- So, try a git add command, with nothing more:

```
[4] ~/grocery (master)$ git add
Nothing specified, nothing added.Maybe you wanted to
say 'git add .'?
```

Staging area, working tree, and HEAD commit

- Git version 1.x:

	New files	Modified files	Deleted files	
git add -A	yes	yes	yes	Stage all (new, modified, deleted) files
git add .	yes	yes	no	Stage new and modified files only
git add -u	no	yes	yes	Stage modified and deleted files only

Staging area, working tree, and HEAD commit

- Git version 2.x:

	New files	Modified files	Deleted files	
git add -A	yes	yes	yes	Stage all (new, modified, deleted) files
git add .	yes	yes	yes	Stage all (new, modified, deleted) files
git add --ignore-removal .	yes	yes	no	Stage new and modified files only
git add -u	no	yes	yes	Stage modified and deleted files only

Staging area, working tree, and HEAD commit

- Another basic usage is to specify the file we want to add; let's give it a try:

```
[5] ~/grocery (master)$ git add shoppingList.txt
```

Staging area, working tree, and HEAD commit

- Okay, time to look back at our repository; go with a git status now:

[6] ~/grocery (master)$ git statusOn branch master
Changes to be committed: (use "git reset HEAD <file>..." to unstage)modified: shoppingList.txt

Staging area, working tree, and HEAD commit

- For now, leave things how they are, and do a commit:

[7] ~/grocery (master)$ git commit -m "Add a peach“
[master 603b9d1] Add a peach 1 file changed, 1 insertion(+)

- Check the status:

[8] ~/grocery (master)$ git statusOn branch master
nothing to commit, working tree clean

Staging area, working tree, and HEAD commit

- So, follow me and make things more interesting; add an onion to the shopping list and then add it to the staging area, and then add a garlic and see what happens:

```
[9] ~/grocery (master)$ echo "onion" >> shoppingList.txt
[10] ~/grocery (master)$ git add shoppingList.txt
[11] ~/grocery (master)$ echo "garlic" >> shoppingList.txt
[12] ~/grocery (master)$ git statusOn branch master
Changes to be committed:  (use "git reset HEAD <file>..." to
unstage)modified:   shoppingList.txtChanges not staged for commit:  (use
"git add <file>..." to update what will be committed)  (use "git checkout --
<file>..." to discard changes in working directory)modified:   shoppingList.txt
```

Staging area, working tree, and HEAD commit

- If you want to see the difference between the working tree version and the staging area one, try to input only the git diff command without any option or argument:

```
[13] ~/grocery (master)$ git diff
diff --git a/shoppingList.txt b/shoppingList.txtindex
f961a4c..20238b5 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -3,3 +3,4 @@ apple orange peach
onion+garlic
```

Staging area, working tree, and HEAD commit

- We have to use the git diff --cached HEAD command:

```
[14] ~/grocery (master)$ git diff --cached HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
175eeef..f961a4c 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -2,3 +2,4 @@ banana apple
orange peach+onion
```

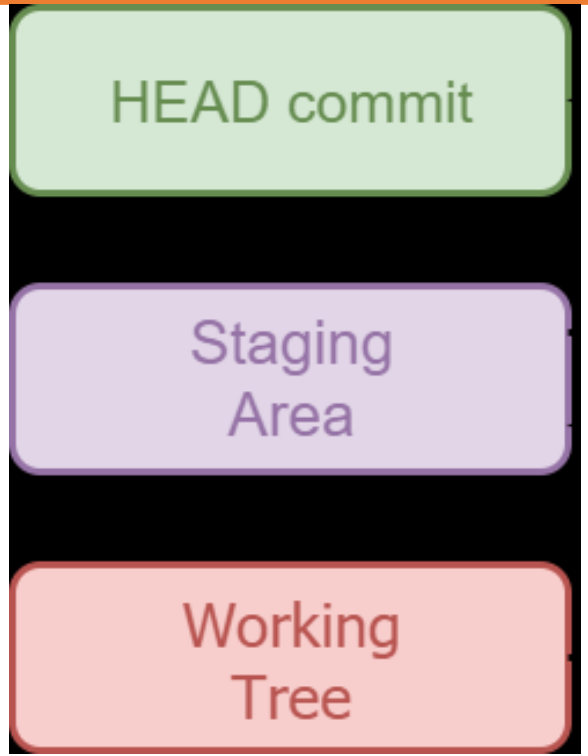
Staging area, working tree, and HEAD commit

- The last experiment that we can do is compare the HEAD version with the working tree one; let's do it with a git diff HEAD:

```
[15] ~/grocery (master)$ git diff HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
175eeef..20238b5 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -2,3 +2,5 @@ banana apple
orange peach+onion+garlic
```

Staging area, working tree, and HEAD commit

- The following figure draws three areas of Git:



Staging area, working tree, and HEAD commit

- Check the repository current status:

[16] ~/grocery (master)$ git statusOn branch master Changes to be committed: (use "git reset HEAD <file>..." to unstage)modified: shoppingList.txtChanges not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory)modified: shoppingList.txt

Staging area, working tree, and HEAD commit

- This is the actual situation, remember? We have an onion in the staging area and a garlic more in the working tree.

- Now go with a git reset HEAD:

```
[17] ~/grocery (master)$ git reset HEAD
shoppingList.txtUnstaged changes after reset:M   shoppingList.txt
```

Staging area, working tree, and HEAD commit

- Well, time to verify what happened:

```
[18] ~/grocery (master)$ git status
On branch master
Changes not staged for commit:

(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working
directory)modified:   shoppingList.txt no changes added to
commit (use "git add" and/or "git commit -a")
```

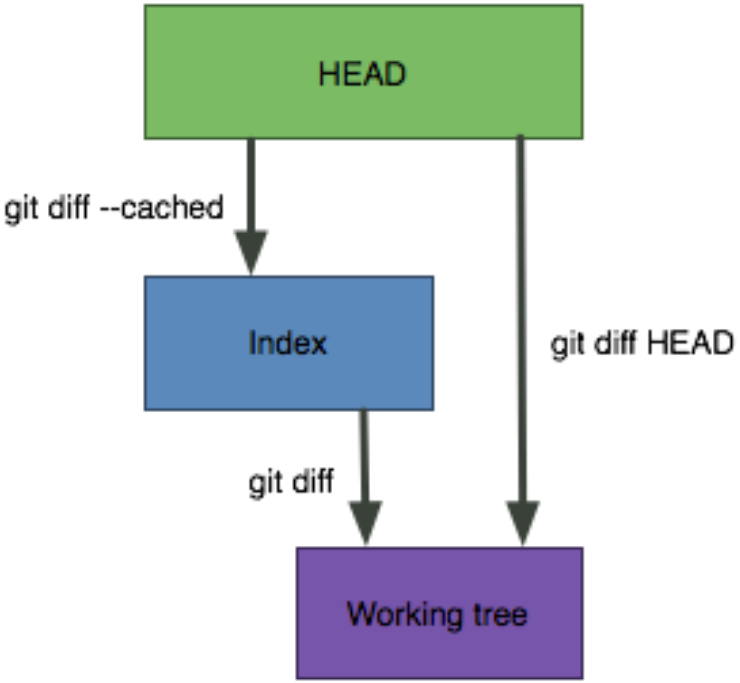
Staging area, working tree, and HEAD commit

- Let's verify this using the git diff command:

```
[19] ~/grocery (master)$ git diff
diff --git a/shoppingList.txt b/shoppingList.txtindex
175eeef..20238b5 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -2,3 +2,5 @@ banana apple
orange peach+onion+garlic
```

Staging area, working tree, and HEAD commit

- The following figure shows a quick summary of git diff different behaviors:



Staging area, working tree, and HEAD commit

- The command for that is git checkout -- <file>, as Git gently reminds in the git status output message. Give it a try:

```
[20] ~/grocery (master)$ git checkout -- shoppingList.txt
```

- Check the status:

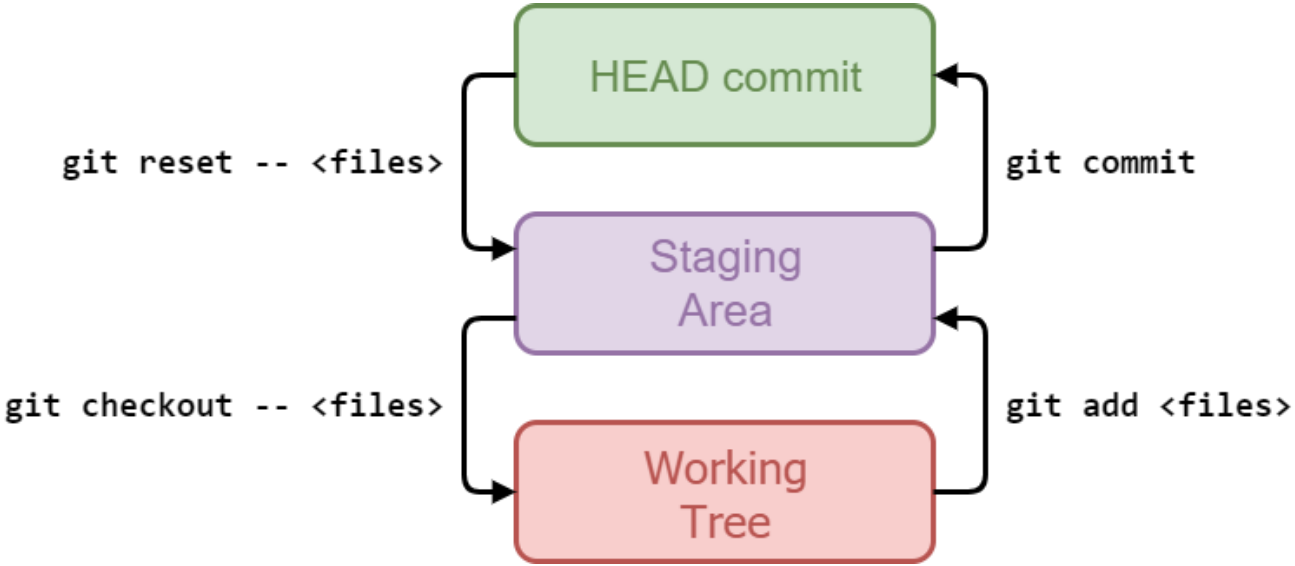
```
[21] ~/grocery (master)$ git statusOn branch master
nothing to commit, working tree clean
```

- Check the content of the file:

```
[22] ~/grocery (master)$ cat shoppingList.txt
bananaappleorangepeach
```

Staging area, working tree, and HEAD commit

- The following figure summarizes the commands to move changes between those three areas:



Staging area, working tree, and HEAD commit

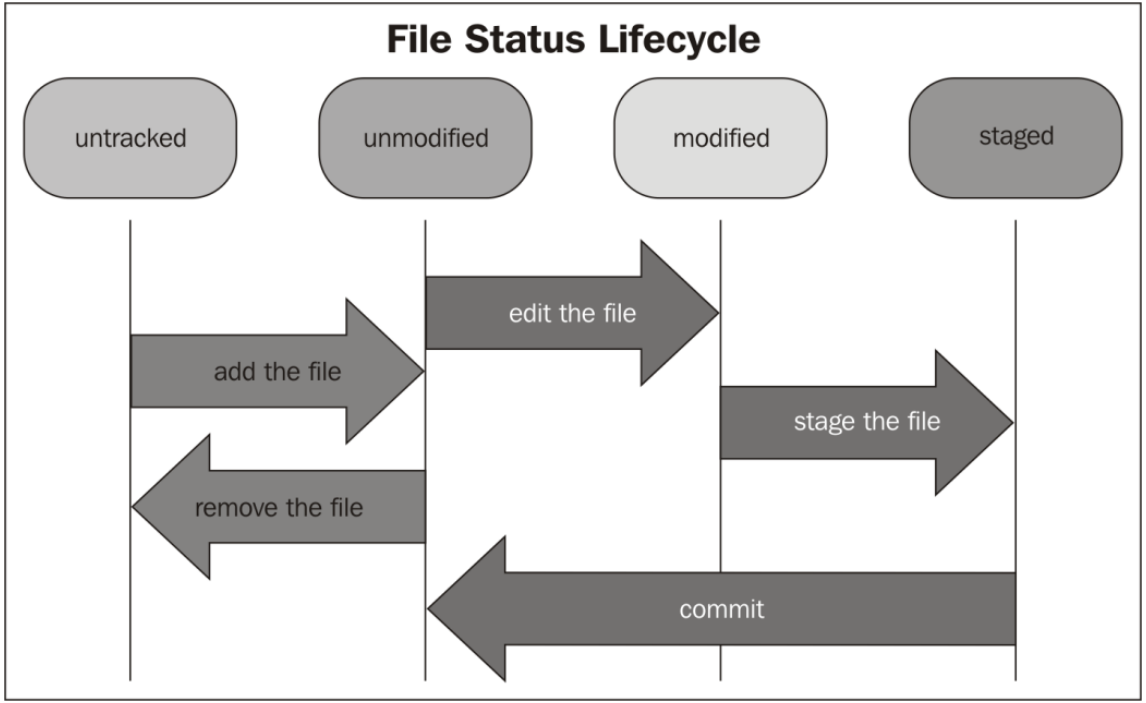
File status lifecycle

- In a Git repository, files pass through some different states.
- When you first create a file in the working tree, Git notices it and tells you there's a new untracked file; let's try to create a new file.txt in our grocery repository and see the output of git status:

[23] ~/grocery (master)$ git statusOn branch master
Untracked files: (use "git add <file>..." to include in what will be committed)file.txtnothing added to commit but untracked files present (use "git add" to track)

Staging area, working tree, and HEAD commit

- The following figure summarizes these states:



Staging area, working tree, and HEAD commit

All you need to know about checkout and reset

- First of all, we need to do some housekeeping.
- Go back to the grocery repository and clean up the working tree; double-check that you are in the master branch, and then do a git reset --hard master:

```
[24] ~/grocery (master)$ git reset --hard master
HEAD is now at 603b9d1 Add a peach
```

Staging area, working tree, and HEAD commit

- Then, delete the bug branch we created some time ago; the command to delete a branch is again the git branch command, this time followed by a -d option and then the branch name:

```
[25] ~/grocery (master)$ git branch -d bug
error: The branch 'bug' is not fully merged.If you are
sure you want to delete it, run 'git branch -D bug'.
```

Staging area, working tree, and HEAD commit

- No problem, we don't need that commit; so, use the capital -D option to force the deletion:

```
[26] ~/grocery (master)$ git branch -D bug
Deleted branch bug (was 07b1858).
```

- Okay, now we are done, and the repository is in good shape, as the git log command shows:

```
[27] ~/grocery (master)$ git log --oneline --graph --decorate –all
* 603b9d1 (HEAD -> master) Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list
```

Staging area, working tree, and HEAD commit

- Git checkout overwrites all the tree areas
- Now switch to the melons branch using the git checkout command:

[28] ~/grocery (master)$ git checkout melons
Switched to branch 'melons'

- Check the log:

[29] ~/grocery (melons)$ git log --oneline --graph --decorate –all
* 603b9d1 (master) Add a peach| * a8c6219 (HEAD -> melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Staging area, working tree, and HEAD commit

- We can try it; add a potato to the shopping list file:

```
[30] ~/grocery (melons)$ echo "potato" >> shoppingList.txt
```

- Then checkout master:

```
[31] ~/grocery (melons)$ git checkout master
error: Your local changes to the following files would be
overwritten by checkout:shoppingList.txtPlease commit
your changes or stash them before you switch
branches.Aborting
```

Staging area, working tree, and HEAD commit

- To avoid messing up our repo again, go into a detached HEAD state, so at the end it will be easier to throw all the things away.
- To do this, checkout directly the penultimate commit on the master branch:

[32] ~/grocery (master)$ git checkout HEAD ~1Note: checking out 'HEAD~1'.You are in 'detached HEAD' state. You can look around, make experimentalchanges and commit them, and you can discard any commits you make in thisstate without impacting any branches by performing another checkout.If you want to create a new branch to retain commits you create, you maydo so (now or later) by using -b with the checkout command again. Example: git checkout -b <new-branch-name>HEAD is now at 0e8b5cf... Add an orange

Staging area, working tree, and HEAD commit

- Now just replicate the onion and garlic situation we used before: append an onion to the file and add it to the staging area, and then add a garlic:

```
[34] ~/grocery ((0e8b5cf...))$ echo "onion" >> shoppingList.txt
[35] ~/grocery ((0e8b5cf...))$ git add shoppingList.txt
[36] ~/grocery ((0e8b5cf...))$ echo "garlic" >> shoppingList.txt
[37] ~/grocery ((0e8b5cf...))$ git status
HEAD detached at 0e8b5cfChanges to be committed:  (use "git
reset HEAD <file>..." to unstage)modified:
shoppingList.txtChanges not staged for commit:  (use "git add
<file>..." to update what will be committed)  (use "git checkout --
<file>..." to discard changes in working directory)modified:
shoppingList.txt
```

Staging area, working tree, and HEAD commit

- Now use the git diff command to be sure we are in the situation we desire; check the differences with the staging area:

```
[38] ~/grocery ((0e8b5cf...))$ git diff --cached HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
edc9072..063aa2f 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -1,3 +1,4 @@ banana apple
orange+onion
```

Staging area, working tree, and HEAD commit

- Check the differences between the working tree and HEAD commit:

```
[39] ~/grocery ((0e8b5cf...))$ git diff HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
edc9072..93dcf0e 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -1,3 +1,5 @@ banana apple
orange+onion+garlic
```

Staging area, working tree, and HEAD commit

- Now try to do a soft reset to the master branch, with the git reset --soft master command:

[40] ~/grocery ((0e8b5cf...))$ git reset --soft master

- Diff to the staging area:

[41] ~/grocery ((603b9d1...))$ git diff --cached HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex 175eeef..063aa2f 100644--- a/shoppingList.txt+++ b/shoppingList.txt@@ -1,4 +1,4 @@ banana apple orange-peach+onion

Staging area, working tree, and HEAD commit

- The same is if you compare the HEAD commit with a working tree:

```
[42] ~/grocery ((603b9d1...))$ git diff HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
175eeef..93dcf0e 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -1,4 +1,5 @@ banana apple
orange-peach+onion+garlic
```

Staging area, working tree, and HEAD commit

- Another option is the mixed reset; you can do it using the --mixed option (or simply using no options, as this is the default):

```
[43] ~/grocery ((603b9d1...))$ git reset --mixed master
Unstaged changes after reset:M   shoppingList.txt
```

- Okay, there's something different here: Git tells us about unstaged changes.
- In fact, the --mixed option makes Git overwrite even the staging area, not only the HEAD commit.
- If you check differences between the HEAD commit and staging area with git diff, you will see that there are no differences:

```
[44] ~/grocery ((603b9d1...))$ git diff --cached HEAD
```

Staging area, working tree, and HEAD commit

- Instead, differences arise between the HEAD commit and working tree:

```
[45] ~/grocery ((603b9d1...))$ git diff HEAD
diff --git a/shoppingList.txt b/shoppingList.txtindex
175eeef..93dcf0e 100644--- a/shoppingList.txt+++
b/shoppingList.txt@@ -1,4 +1,5 @@ banana apple
orange-peach+onion+garlic
```

Staging area, working tree, and HEAD commit

- At this point, you can presume what is the purpose of the --hard option: it overwrites all the three areas:

```
[46] ~/grocery ((603b9d1...))$ git reset --hard master
HEAD is now at 603b9d1 Add a peach[47] ~/grocery
((603b9d1...))$ git diff --cached HEAD[48] ~/grocery
((603b9d1...))$ git diff HEAD
```

Staging area, working tree, and HEAD commit

- We are done.

- Now we know a little more about both the git checkout and git reset command; but before leaving, go back in a non-detached HEAD state, checking out the master branch:

```
[49] ~/grocery ((603b9d1...))$ git checkout master
Switched to branch 'master'
```

Rebasing

Basically, with git rebase you rewrite history; with this statement, I mean you can use rebase command to achieve the following:

- Combine two or more commits into a new one
- Discard a previous commit you did
- Change the starting point of a branch, split it, and much more

Rebasing

Reassembling commits

- Suppose we erroneously added half a grape in the shoppingList.txt file, then the other half, but at the end we want to have only one commit for the entire grape; follow me with these steps.

- Add a gr to the shopping list file:

```
[1] ~/grocery (master)$ echo -n "gr" >> shoppingList.txt
```

Rebasing

- The -n option is for not adding a new line.
- Cat the file to be sure:

[2] ~/grocery (master)$ cat shoppingList.txt
bananaappleorangepeachgr

- Now perform the first commit:

[3] ~/grocery (master)$ git commit -am "Add half a grape"[master edac12c] Add half a grape 1 file changed, 1 insertion(+)

Rebasing

- Okay, we have a commit with half a grape. Go on and add the other half, ape:

[4] ~/grocery (master)$ echo -n "ape" >> shoppingList.txt

- Check the file:

[5] ~/grocery (master)$ cat shoppingList.txt
bananaappleorangepeachgrape

- Perform the second commit:

[6] ~/grocery (master)$ git commit -am "Add the other half of the grape“
[master 4142ad9] Add the other half of the grape 1 file changed, 1 insertion(+), 1 deletion(-)

Rebasing

- Check the log:

[7] ~/grocery (master)$ git log --oneline --graph --decorate –all
* 4142ad9 (HEAD -> master) Add the other half of the grape* edac12c Add half a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Rebasing

This is a screenshot of the console:

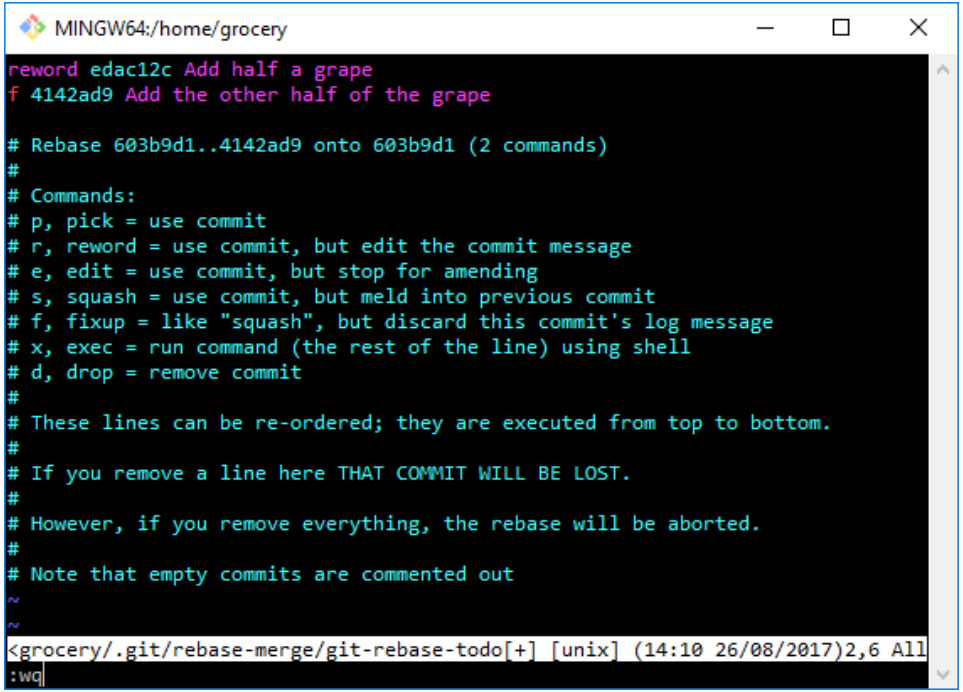
MINGW64:/home/grocery

```
pick edac12c Add half a grape
pick 4142ad9 Add the other half of the grape

# Rebase 603b9d1..4142ad9 onto 603b9d1 (2 commands)
#
# 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
# d, drop = remove commit
#
# 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
~
~
~/grocery/.git/rebase-merge/git-rebase-todo [unix] (14:10 26/08/2017)  1,1 All
```

Rebasing

- To resolve our issue, I will reword the first commit and then fixup the second; the following is a screenshot of my console:



Rebasing

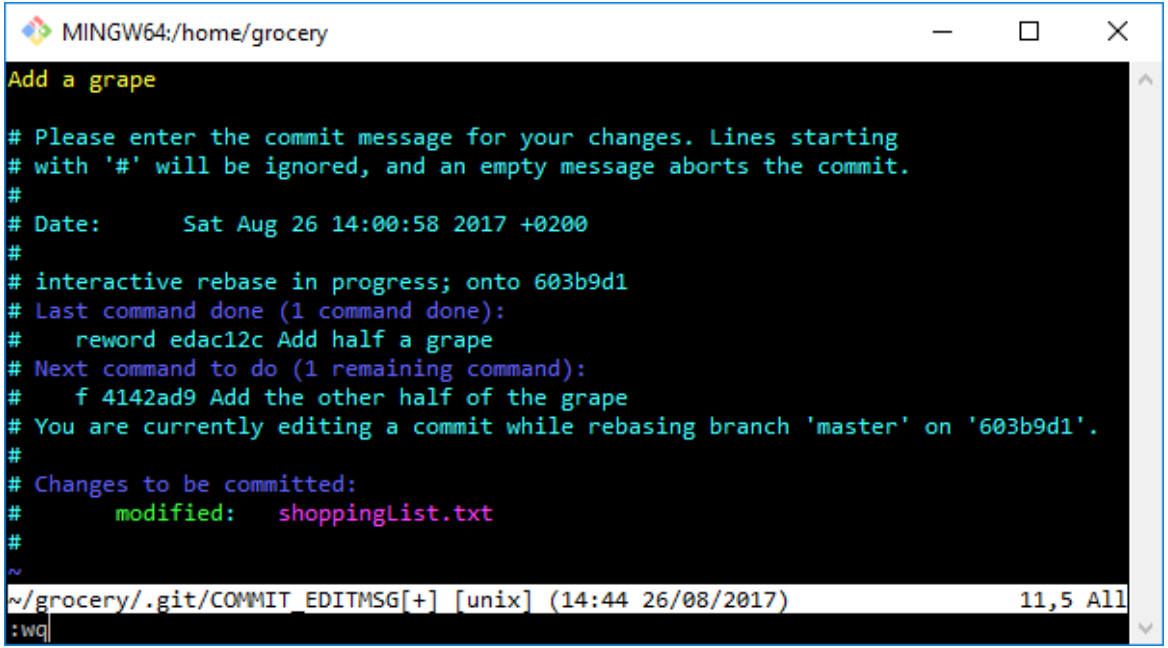
MINGW64:/home/grocery

```
Add half a grape

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sat Aug 26 14:00:58 2017 +0200
#
# interactive rebase in progress; onto 603b9d1
# Last command done (1 command done):
#    reword edac12c Add half a grape
# Next command to do (1 remaining command):
#    f 4142ad9 Add the other half of the grape
# You are currently editing a commit while rebasing branch 'master' on '603b9d1'.
#
# Changes to be committed:
#       modified:   shoppingList.txt
#
~
~/grocery/.git/COMMIT_EDITMSG [unix] (14:44 26/08/2017)                1,1 All
```

Rebasing

- Now edit the message, and then save and exit, like in the following screenshot:



Rebasing

- This is the final message from Git:

[8] ~/grocery (master)$ git rebase -i HEAD
~2unix2dos: converting file C:/Users/san/Google Drive/Packt/PortableGit/home/grocery/[detached HEAD 53c73dd] Add a grape Date: Sat Aug 26 14:00:58 2017 +0200 1 file changed, 1 insertion(+)Successfully rebased and updated refs/heads/master.

- Take a look at the log:

[9] ~/grocery (master)$ git log --oneline --graph --decorate –all
* 6409527 (HEAD -> master) Add a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Rebasing

- Let's start by creating a new branch that points to commit 0e8b5cf, the orange one:

[1] ~/grocery (master)$ git branch nuts 0e8b5cf

- This time I used the git branch command followed by two arguments, the name of the branch and the commit where to stick the label. As a result, a new nuts branch has been created:

[2] ~/grocery (master)$ git log --oneline --graph --decorate –all
* 6409527 (HEAD -> master) Add a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf (nuts) Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Rebasing

- Move HEAD to the new branch with the git checkout command:

```
[3] ~/grocery (master)$ git checkout nuts
Switched to branch 'nuts'
```

- Okay, now it's time to add a walnut; add it to the shoppingList.txt file:

```
[4] ~/grocery (nuts)$ echo "walnut" >> shoppingList.txt
```

- Then do the commit:

```
[5] ~/grocery (nuts)$ git commit -am "Add a walnut"
[master 3d3ae9c] Add a walnut 1 file changed, 1 insertion(+), 1 deletion(-)
```

Rebasing

Check the log:

[6] ~/grocery (nuts)$ git log --oneline --graph --decorate –all
* 9a52383 (HEAD -> nuts) Add a walnut| * 6409527 (master) Add a grape| * 603b9d1 Add a peach|/| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Rebasing

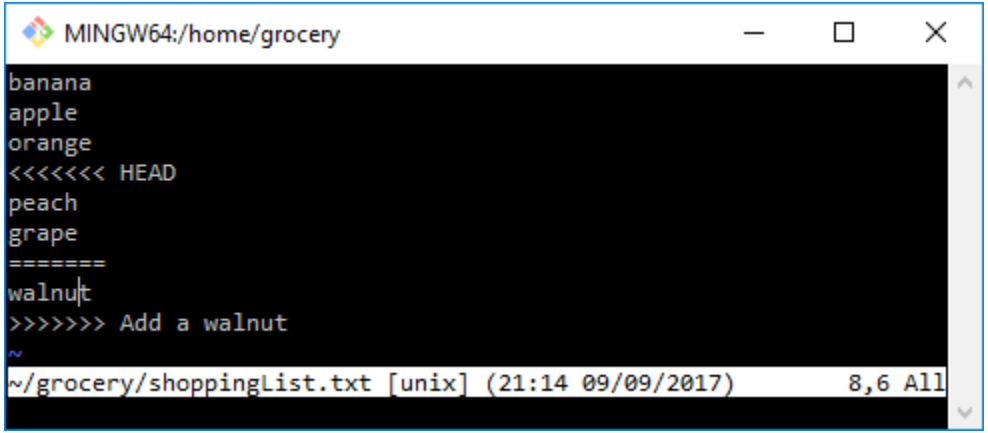
- Let's do it, rebasing the nuts branch on top of master; double-check that you actually are in the nuts branch, as a rebase command basically rebases the current branch (nuts) to the target one, master; so:

[7] ~/grocery (nuts)$ git rebase masterFirst, rewinding head to replay your work on top of it...Applying: Add a walnutUsing index info to reconstruct a base tree...M shoppingList.txtFalling back to patching base and 3-way merge...Auto-merging shoppingList.txtCONFLICT (content): Merge conflict in shoppingList.txtPatch failed at 0001 Add a walnutThe copy of the patch that failed is found in: .git/rebase-apply/patchWhen you have resolved this problem, run "git rebase --continue".If you prefer to skip this patch, run "git rebase --skip" instead.To check out the original branch and stop rebasing, run "git rebase --abort".error: Failed to merge in the changes.

Rebasing

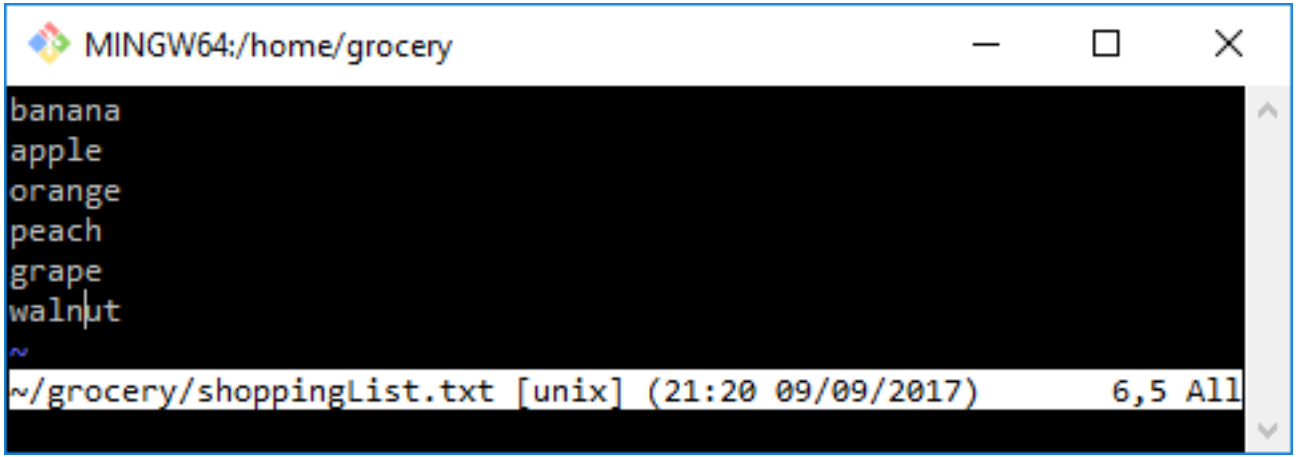
- Now, back to our repository; if you open the file with Vim, you can see the generated conflict:

[8] ~/grocery (nuts|REBASE 1/1)$ vi shoppingList.txt



Rebasing

- I will fix it adding the walnut at the end of the file:



Rebasing

- Now, the next step is to git add the shoppingList.txt file to the staging area, and then go on with the git rebase --continue command, as the previous message suggested:

```
[9] ~/grocery (nuts|REBASE 1/1)$ git add shoppingList.txt
[10] ~/grocery (nuts|REBASE 1/1)$ git rebase –continue
Applying: Add a walnut[11] ~/grocery (nuts)$
```

Rebasing

Now take a look at the repo using git log as usual:

[12] ~/grocery (nuts)$ git log --oneline --graph --decorate –all
* 383d95d (HEAD -> nuts) Add a walnut* 6409527 (master) Add a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Rebasing

- Okay, now to keep the simplest and most compact repository, we cancel the walnut commit and put everything back in place as it was before this little experiment, even removing the nuts branch:

[13] ~/grocery (nuts)$ git reset --hard HEAD^
HEAD is now at 6409527 Add a grape[14] ~/grocery (nuts)$ git checkout masterSwitched to branch 'master'[15] ~/grocery (master)$ git branch -d nutsDeleted branch nuts (was 6409527).

Merging branches

In Git, merging two (or more!) branches is the act of making their personal history meet each other. When they meet, two things can happen:

- Files in their tip commit are different, so some conflict will rise
- Files do not conflict
- Commits of the target branch are directly behind commits of the branch we are merging, so a fast-forward will happen

Merging branches

- Let's give it a try.
- We can try to merge the melons branch into the master one; to do so, you have to check out the target branch, master in this case, and then fire a git merge <branch name> command; as I'm already on the master branch, I go straight with the merge command:

```
[1] ~/grocery (master)$ git merge melons
Auto-merging shoppingList.txtCONFLICT (content): Merge conflict in shoppingList.txtAutomatic merge failed; fix conflicts and then commit the result.
```

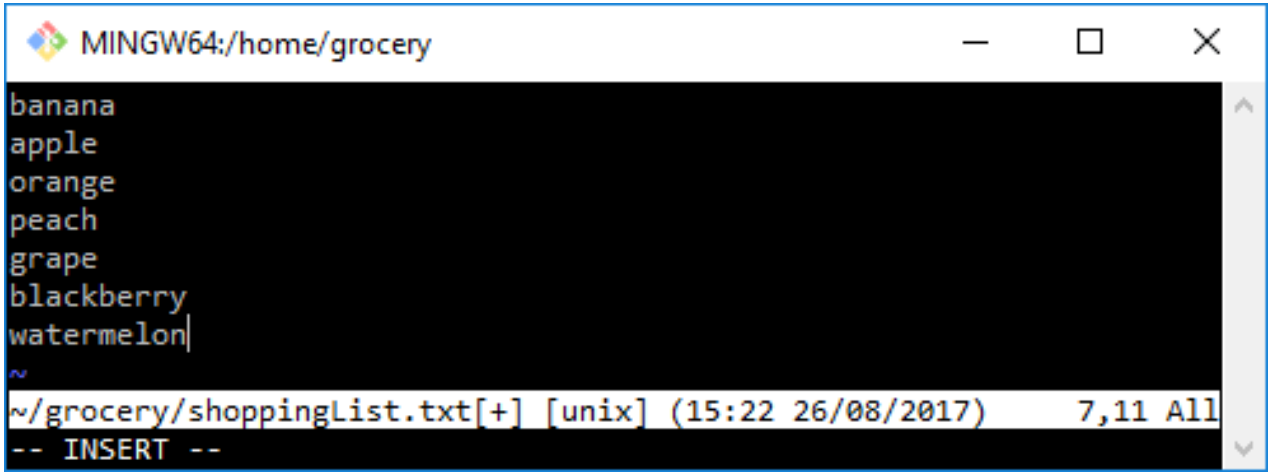
Merging branches

- See the conflict with git diff:

```
[2] ~/grocery (master|MERGING)$ git diff
diff --cc shoppingList.txtindex 862debc,7786024..0000000--
- a/shoppingList.txt+++ b/shoppingList.txt@@@ -1,5 -1,5
+1,10 @@@  banana  apple  orange++<<<<<<< HEAD
+peach- grape++grape++=======+ blackberry+
watermelon++>>>>>>> melons
```

Merging branches

- I will edit the file enqueuing blackberry and watermelon after peach and grape, as per the following screenshot:



Merging branches

- After saving the file, add it to the staging area and then commit:

```
[3] ~/grocery (master|MERGING)$ git add shoppingList.txt
[4] ~/grocery (master|MERGING)$ git commit -m "Merged melons branch into master“
[master e18a921] Merged melons branch into master
```

Merging branches

- Now take a look at the log:

[5] ~/grocery (master)$ git log --oneline --graph --decorate –all
* e18a921 (HEAD -> master) Merged melons branch into master|\| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry* | 6409527 Add a grape* | 603b9d1 Add a peach|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Merging branches

- Suggestion: look at the merge commit with git cat-file -p:

```
[6] ~/grocery (master)$ git cat-file -p HEAD
tree 2916dd995ee356351c9b49a5071051575c070e5fparent
6409527a1f06d0bbe680d461666ef8b137ac7135parent
a8c62190fb1c54d1034db78a87562733a6e3629cauthor Ernesto
Lee <socrates73@gmail.com> 1503754221 +0200committer
Ernesto Lee <socrates73@gmail.com> 1503754221
+0200Merged melons branch into master
```

Merging branches

Fast forwarding

- A merge not always generates a new commit; to test this case, try to merge the melons branch into a berries one:

[7] ~/grocery (master)$ git checkout berries
Switched to branch 'berries’
[8] ~/grocery (berries)$ git merge melons
Updating ef6c382..a8c6219Fast-forward shoppingList.txt | 1 + 1 file changed, 1 insertion(+)
[9] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* e18a921 (master) Merged melons branch into master|\| * a8c6219 (HEAD -> berries, melons) Add a watermelon| * ef6c382 Add a blackberry* | 6409527 Add a grape* | 603b9d1 Add a peach|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Merging branches

- Move back the berries branch where it was using git reset:

```
[10] ~/grocery (berries)$ git reset --hard HEAD^
HEAD is now at ef6c382 Add a blackberry
[11] ~/grocery (berries)$ git log --oneline --graph --decorate –all
*   e18a921 (master) Merged melons branch into master|\| *
a8c6219 (melons) Add a watermelon| * ef6c382 (HEAD ->
berries) Add a blackberry* | 6409527 Add a grape* | 603b9d1
Add a peach|/* 0e8b5cf Add an orange* e4a5e7b Add an
apple* a57d783 Add a banana to the shopping list
```

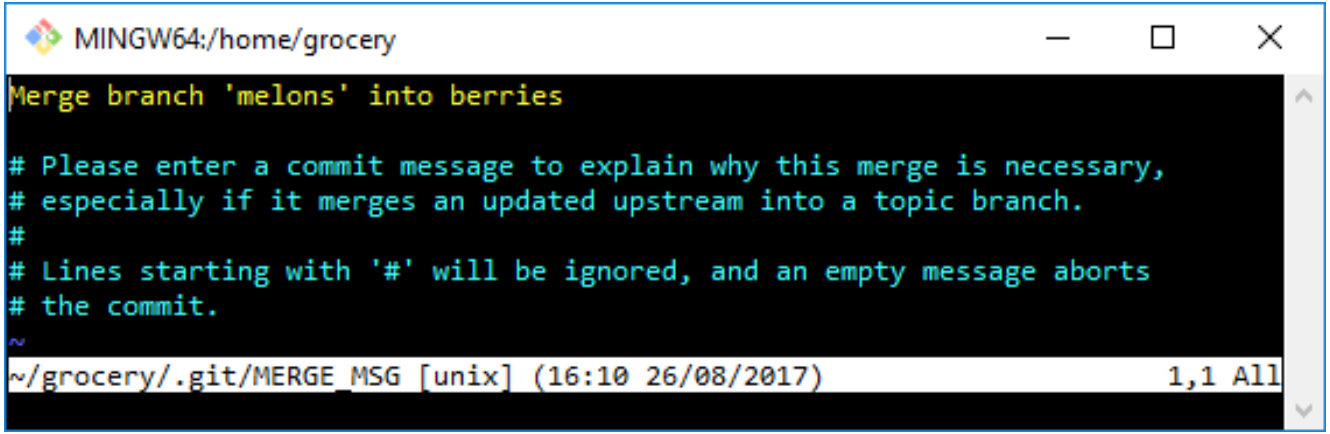
Merging branches

- We have just undone a merge, did you realize it?
- Okay, now do the merge again with the --no-ff option:

```
[12] ~/grocery (berries)$ git merge --no-ff melons
```

Merging branches

- Git will now open your default editor to allow you to specify a commit message, as shown in the following screenshot:



Merging branches

- Accept the default message, save and exit:

[13] ~/grocery (berries)Merge made by the 'recursive' strategy.--all shoppingList.txt | 1 + 1 file changed, 1 insertion(+)

Merging branches

- Now a git log:

[14] ~/grocery (berries)$ git log --oneline --graph --decorate –all
* cb912b2 (HEAD -> berries) Merge branch 'melons' into berries|\| | * e18a921 (master) Merged melons branch into master| | |\| | |/| |/|| * | a8c6219 (melons) Add a watermelon|/ /* | ef6c382 Add a blackberry| * 6409527 Add a grape| * 603b9d1 Add a peach|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Merging branches

- We are done with these experiments; anyway, I want to undo this merge, because I want to keep the repository as simple as possible to allow you to better understand the exercise we do together; go with a git reset --hard HEAD^:

[15] ~/grocery (berries)$ git reset --hard HEAD^
HEAD is now at ef6c382 Add a blackberry[16] ~/grocery (berries)$ git log --oneline --graph --decorate --all* e18a921 (master) Merged melons branch into master|\| * a8c6219 (melons) Add a watermelon| * ef6c382 (HEAD -> berries) Add a blackberry* | 6409527 Add a grape* | 603b9d1 Add a peach|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Merging branches

Okay, now undo even the past merge we did on the master branch:

[17] ~/grocery (master)$ git reset --hard HEAD^
HEAD is now at 6409527 Add a grape[18] ~/grocery (master)$ git log --oneline --graph --decorate --all* 6409527 (HEAD -> master) Add a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Cherry picking

- Let's play with it a little bit.
- Assume you want to pick the blackberry from the berries branch, and then apply it into the master branch; this is the way:

```
[1] ~/grocery (master)$ git cherry-pick
ef6c382error: could not apply ef6c382... Add a
blackberryhint: after resolving the conflicts, mark the
corrected pathshint: with 'git add <paths>' or 'git rm
<paths>'hint: and commit the result with 'git commit'
```

Cherry picking

- For the argument, you usually specify the hash of the commit you want to pick; in this case, as that commit is referenced even by the berries branch label, doing a git cherry-pick berries would have been the same.
- Okay, the cherry pick raised a conflict, of course:

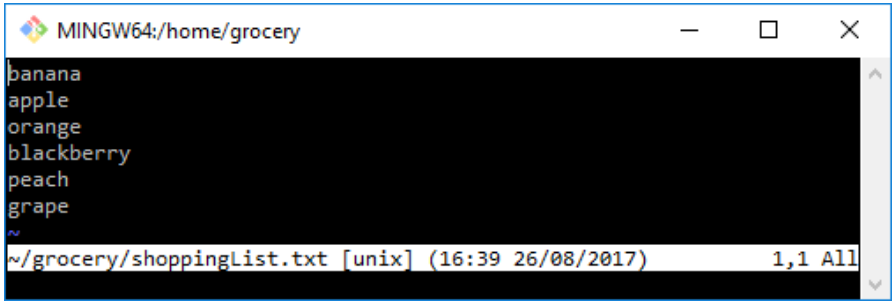
[2] ~/grocery (master|CHERRY-PICKING)$ git diff
diff --cc shoppingList.txtindex 862debc,b05b25f..0000000---
a/shoppingList.txt+++ b/shoppingList.txt@@@ -1,5 -1,4 +1,9
@@@ banana apple orange++<<<<<<< HEAD +peach-
grape++grape++=======+ blackberry++>>>>>>> ef6c382... Add
a blackberry

Cherry picking

- The fourth line of both the shoppingList.txt file versions has been modified with different fruits.
- Resolve the conflict and then add a commit:

[3] ~/grocery (master|CHERRY-PICKING)$ vi shoppingList.txt

- The following is a screenshot of my Vim console, and the files are arranged as I like:



Cherry picking

[4] ~/grocery (master|CHERRY-PICKING)$ git add shoppingList.txt
[5] ~/grocery (master|CHERRY-PICKING)$ git statusOn branch master
You are currently cherry-picking commit ef6c382. (all conflicts fixed: run "git cherry-pick --continue") (use "git cherry-pick --abort" to cancel the cherry-pick operation)Changes to be committed:modified: shoppingList.txt

Cherry picking

- Now go on and commit:

[6] ~/grocery (master)$ git commit -m "Add a cherry-picked blackberry“

On branch masternothing to commit, working tree clean[7] ~/grocery (master)$ git log --oneline --graph --decorate --all* 99dd471 (HEAD -> master) Add a cherry-picked blackberry* 6409527 Add a grape* 603b9d1 Add a peach| * a8c6219 (melons) Add a watermelon| * ef6c382 (berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Cherry picking

MINGW64:/home/grocery

```
Add a blackberry

(cherry picked from commit ef6c3821fdfa92b90bc9fc444befffc5326f4228)

# Conflicts:
#       shoppingList.txt
#
# It looks like you may be committing a cherry-pick.
# If this is not correct, please remove the file
#       .git/CHERRY_PICK_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Aug 25 13:24:38 2017 +0200
#
# On branch master
# You are currently cherry-picking commit ef6c382.
#
# Changes to be committed:
#       modified:   shoppingList.txt
#
~
~
~/grocery/.git/COMMIT_EDITMSG [unix] (16:56 26/08/2017)          8,1 All
```

Summary

- This has been a very long lesson, I know.
- But now I think you know all you need to work proficiently with Git, at least in your own local repository.
- You know about working tree, staging area, and HEAD commit; you know about references as branches and HEAD; you know how to merge rebase, and cherry pick; and finally, you know how Git works under the hood, and this will help you from here on out.

3. Git Fundamentals - Working Remotely

LEARNING VOYAGE

Git Fundamentals - Working Remotely

In this lesson, we finally start to work in a distributed manner, using remote servers as a contact point for different developers. These are the main topics we will focus on:

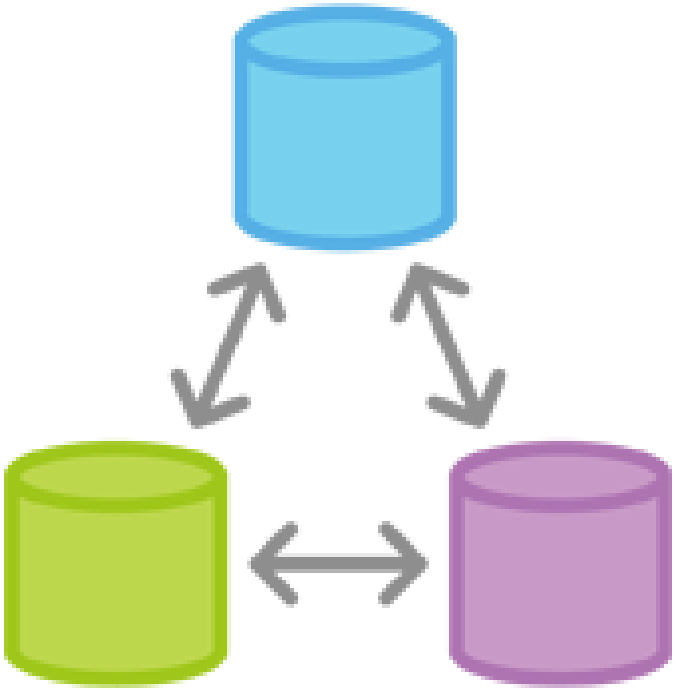
- Dealing with remotes
- Cloning a remote repository
- Working with online hosting services, such as GitHub

Working with remotes

- Git is a tool for versioning files, as you know, but it has been built with collaboration in mind.

- In 2005, Linus Torvalds had the need for a light and efficient tool to share the Linux kernel code, allowing him and hundreds of other people to work on it without going crazy; the pragmatism that guided its development gave us a very robust layer for sharing data among computers, without the need of a central server.

Working with remotes



Working with remotes

Clone a local repository

- Create a new folder on your disk to clone our grocery repository:

[1] ~$ mkdir grocery-cloned

- Then clone the grocery repository using the git clone command:

[2] ~$ cd grocery-cloned[3] ~/grocery-cloned$ git clone ~/grocery .Cloning into '.'...

Working with remotes

- Now, go directly to the point with a git log command:

[4] ~/grocery-cloned (master)$ git log --oneline --graph --decorate –all
* 6409527 (HEAD -> master, origin/master, origin/HEAD) Add a grape* 603b9d1 Add a peach| * a8c6219 (origin/melons) Add a watermelon| * ef6c382 (origin/berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Working with remotes

- But don't worry: a local branch in which to work locally can be created by simply checking it out:

```
[5] ~/grocery-cloned (master)$ git checkout berriesBranch
berries set up to track remote branch berries from
origin.Switched to a new branch 'berries'
```

Working with remotes

- Now, look at the log again:

[6] ~/grocery-cloned (berries)$ git log --oneline --graph --decorate --all* 6409527 (origin/master, origin/HEAD, master) Add a grape* 603b9d1 Add a peach| * a8c6219 (origin/melons) Add a watermelon| * ef6c382 (HEAD -> berries, origin/berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Working with remotes

- Let's try:

[7] ~/grocery-cloned (berries)$ echo "blueberry" >> shoppingList.txt[8] ~/grocery-cloned (berries)$ git commit -am "Add a blueberry"[berries ab9f231] Add a blueberryCommitter: Santacroce Ferdinando <san@intre.it>

Working with remotes

- You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"git config --global user.email you@example.com

- After doing this, you may fix the identity used for this commit with the following code:

git commit --amend --reset-author1 file changed, 1 insertion(+)

Working with remotes

OK, let's see what happened:

[9] ~/grocery-cloned (berries)$ git log --oneline --graph --decorate --all* ab9f231 (HEAD -> berries) Add a blueberry| * 6409527 (origin/master, origin/HEAD, master) Add a grape| * 603b9d1 Add a peach| | * a8c6219 (origin/melons) Add a watermelon| |/|/|* | ef6c382 (origin/berries) Add a blackberry|/* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Working with remotes

- Now, we will try to push the modifications in the berries branch to the origin; the command is git push, followed by the name of the remote and the target branch:

```
[10] ~/grocery-cloned (berries)$ git push origin berriesCounting objects: 3, done.Delta compression using up to 8 threads.Compressing objects: 100% (2/2), done.Writing objects: 100% (3/3), 323 bytes | 0 bytes/s, done.Total 3 (delta 0), reused 0 (delta 0)To C:/Users/san/Google Drive/Packt/PortableGit/home/grocery   ef6c382..ab9f231  berries -> berries
```

Working with remotes

- Now, we obviously want to see if, in the remote repository, there is a new commit in the berries branch; so, open the grocery folder in a new console and do git log:

[11] ~$ cd grocery[12] ~/grocery (master)$ git log --oneline --graph --decorate --all* ab9f231 (berries) Add a blueberry| * 6409527 (HEAD -> master) Add a grape| * 603b9d1 Add a peach| | * a8c6219 (melons) Add a watermelon| |/|/|* | ef6c382 Add a blackberry|/

* 0e8b5cf Add an orange* e4a5e7b Add an apple* a57d783 Add a banana to the shopping list

Working with remotes

Getting remote commits with git pull

- Now, it's time to experiment the inverse: retrieving updates from the remote repository and applying them to our local copy.

- So, make a new commit in the grocery repository, and then, we will download it into the grocery-cloned one:

```
[13] ~/grocery (master)$ printf "\r\n" >> shoppingList.txt
```

Working with remotes

- I firstly need to create a new line, because due to the previous grape rebase, we ended having the shoppinList.txt file with no new line at the end, as echo "" >> <file> usually does:

[14] ~/grocery (master)$ echo "apricot" >> shoppingList.txt[15] ~/grocery (master)$ git commit -am "Add an apricot"[master 741ed56] Add an apricot 1 file changed, 2 insertions(+), 1 deletion(-)

Working with remotes

- Let's try git pull for now, then we will try to use git fetch and git merge separately.

- Go back to the grocery-cloned repository, switch to the master branch, and do a git pull:

[16] ~/grocery-cloned (berries)$ git checkout masterYour branch is up-to-date with 'origin/master'.Switched to branch 'master'

Working with remotes

- For now, go with git pull: the command wants you to specify the name of the remote you want to pull from, which is origin in this case, and then the branch you want to merge into your local one, which is master, of course:

[17] ~/grocery-cloned (master)$ git pull origin masterremote: Counting objects: 3, done.remote: Compressing objects: 100% (2/2), done.remote: Total 3 (delta 0), reused 0 (delta 0)Unpacking objects: 100% (3/3), done.From C:/Users/san/Google Drive/Packt/PortableGit/home/grocery * branch master -> FETCH_HEAD 6409527..741ed56 master -> origin/masterUpdating 6409527..741ed56Fast-forward shoppingList.txt | 3 ++- 1 file changed, 2 insertions(+), 1 deletion(-)

Working with remotes

- OK, now I want you to try doing these steps in a separate manner; create the umpteenth new commit in the grocery repository, the master branch:

```
[18] ~/grocery (master)$ echo "plum" >>
shoppingList.txt[19] ~/grocery (master)$ git commit -am
"Add a plum"[master 50851d2] Add a plum1 file changed, 1
insertion(+)
```

Working with remotes

- Now perform a git fetch on grocery-cloned repository:

[20] ~/grocery-cloned (master)$ git fetchremote: Counting objects: 3, done.remote: Compressing objects: 100% (2/2), done.remote: Total 3 (delta 0), reused 0 (delta 0)Unpacking objects: 100% (3/3), done.From C:/Users/san/Google Drive/Packt/PortableGit/home/grocery 741ed56..50851d2 master -> origin/master

Working with remotes

- Do a git status now:

[21] ~/grocery-cloned (master)$ git statusOn branch masterYour branch is behind 'origin/master' by 1 commit, and can be fast-forwarded. (use "git pull" to update your local branch)nothing to commit, working tree clean

Working with remotes

- Now, let's sync with a git merge; to merge a remote branch, we have to specify, other than the branch name, even the remote one, as we did in the git pull command previously:

```
[22] ~/grocery-cloned (master)$ git merge origin
masterUpdating 741ed56..50851d2Fast-forward
shoppingList.txt | 1 + 1 file changed, 1 insertion(+)
```

Working with remotes

How Git keeps track of remotes

- Git stores remote branch labels in a similar way to how it stores the local branches ones; it uses a subfolder in refs for the scope, with the symbolic name we used for the remote, in this case origin, the default one:

[23] ~/grocery-cloned (master)$ ll .git/refs/remotes/origin/total 3drwxr-xr-x 1 san 1049089 0 Aug 27 11:25 ./drwxr-xr-x 1 san 1049089 0 Aug 26 18:19 ../-rw-r--r-- 1 san 1049089 41 Aug 26 18:56 berries-rw-r--r-- 1 san 1049089 32 Aug 26 18:19 HEAD-rw-r--r-- 1 san 1049089 41 Aug 27 11:25 master

Working with a public server on GitHub

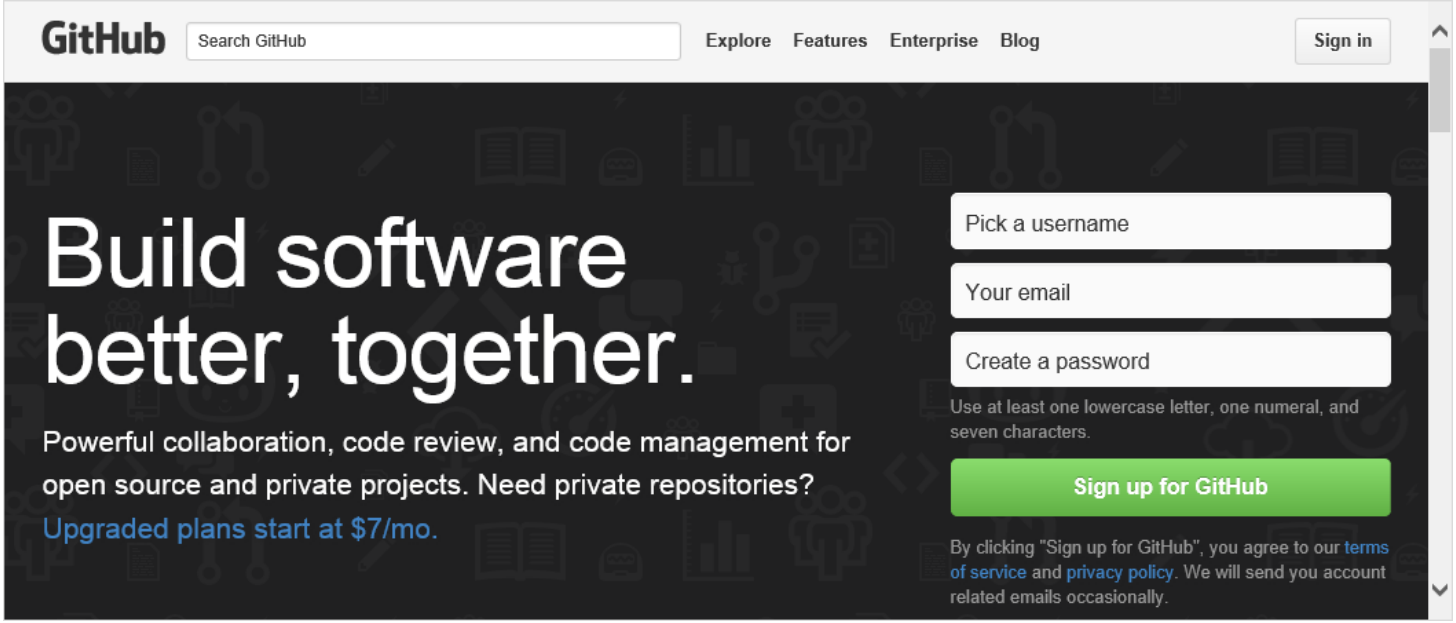
To start working with a public hosted remote, we have to get one.

Today, it is not difficult to achieve; the world has plenty of free online services offering room for Git repositories.

One of the most commonly used is GitHub.

Working with a public server on GitHub

Sign up, filling the textboxes, as per the following image, and provide a username, a password, and your email:



Working with a public server on GitHub

Search GitHub | Explore | Gist | Blog | Help | fsantacroce

Ferdinando Santacroce
fsantacroce

Italy
ferdinando.santacroce@gmail...
http://jesuswasrasta.com
Joined on Nov 23, 2014

0 Followers | 0 Starred | 0 Following

Contributions | Repositories | Public activity | Edit profile

Contributions

Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov
M
W
F

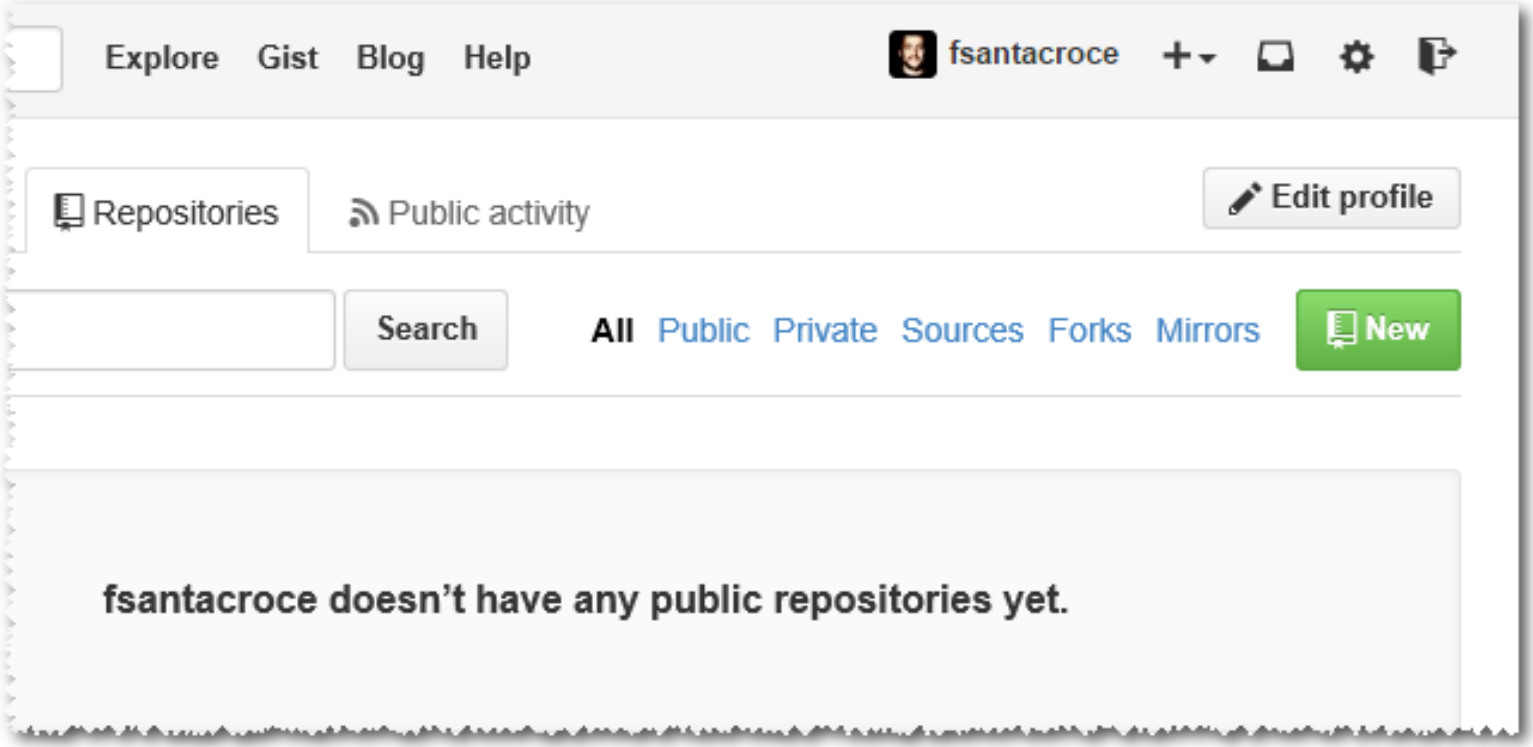
Summary of Pull Requests, issues opened, and commits. Learn more. Less More

This is your **contribution graph**. When you make a commit to a repository, you'll get a for that day. Make more contributions and you'll get a darker green square. Over time, your chart might start looking something like this.

We have a quick guide that will show you how to create your first repository. You'll also make a commit and **earn your first green square!**

Read the Hello World guide

Working with a public server on GitHub



Working with a public server on GitHub

Owner **Repository name**

fsantacroce / Cookbook

Great repository names are short and memorable. Need inspiration? How about **yolo-ironman**.

Description (optional)

This repository contains recipes I like to share with my friends

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

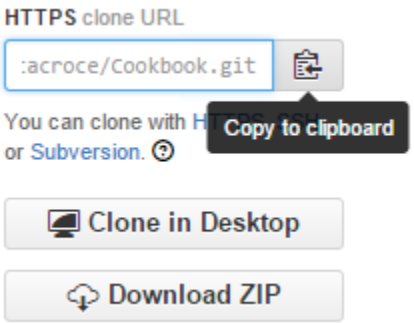
Add .gitignore: **None** | Add a license: **None**

Create repository

Working with a public server on GitHub

Cloning the repository

- Using this command is quite simple; in this case, all we need to know is the URL of the repository to clone.
- The URL is provided by GitHub on the right down part of the repository home page:



Working with a public server on GitHub

- Obviously, the URL of your repository will be different; as you can see, GitHub URLs are composed as follows: https://github.com/<Username>/<RepositoryName>.git:

[1] ~$ git clone https://github.com/fsantacroce/Cookbook.gitCloning into 'Cookbook'...remote: Counting objects: 15, done.remote: Total 15 (delta 0), reused 0 (delta 0), pack-reused 15Unpacking objects: 100% (15/15), done.[2] ~$ cd Cookbook/[3] ~/Cookbook (master)$ lltotal 13drwxr-xr-x 1 san 1049089 0 Aug 27 14:16 ./drwxr-xr-x 1 san 1049089 0 Aug 27 14:16 ../drwxr-xr-x 1 san 1049089 0 Aug 27 14:16 .git/-rw-r--r-- 1 san 1049089 150 Aug 27 14:16 README.md

Working with a public server on GitHub

Uploading modifications to remotes

So, let's try to edit the README.md file and upload modifications to GitHub:

- Edit the README.md file using your preferred editor, adding, for example, a new sentence.
- Add it to the index and then commit.
- Put your commit on the remote repository using the git push command.

Working with a public server on GitHub

- But firstly, set the user and email this time, so Git will not output the message we have seen in the previous lessons:

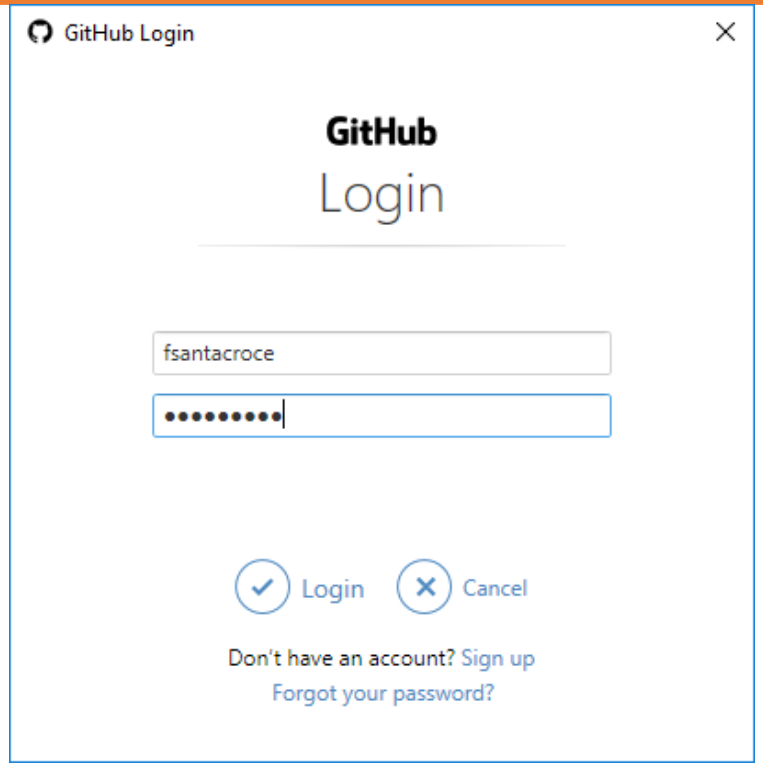
[4] ~/Cookbook (master)$ git config user.name "Ernesto Lee"[5] ~/Cookbook (master)$ git config user.email "socrates73@gmail.com"[6] ~/Cookbook (master)$ vim README.mdAdd a sentence then save and close the editor.[7] ~/Cookbook (master)$ git add README.md[8] ~/Cookbook (master)$ git commit -m "Add a sentence to readme"[master 41bdbe6] Add a sentence to readme 1 file changed, 2 insertions(+)

Working with a public server on GitHub

- Now, try to type git push and press ENTER, without specifying anything else:

[9] ~/Cookbook (master)$ git push

- Here, in my Windows 10 workstation, this window appears:



Working with a public server on GitHub

- Input your credentials, and then press the Login button; after that, Git continues with:

[10] ~/Cookbook (master)$ git pushCounting objects: 3, done.Delta compression using up to 8 threads.Compressing objects: 100% (2/2), done.Writing objects: 100% (3/3), 328 bytes | 0 bytes/s, done.Total 3 (delta 1), reused 0 (delta 0)remote: Resolving deltas: 100% (1/1), completed with 1 local object.To https://github.com/fsantacroce/Cookbook.git e1e7236..41bdbe6 master -> master

Working with a public server on GitHub

This repository | Search | Pull requests | Issues | Marketplace | Explore

fsantacroce / Cookbook

Unwatch 1 | Star 0 | Fork 3

Code | Issues 0 | Pull requests 1 | Projects 0 | Wiki | Settings | Insights

This repository contains recipes I like to share with my friends

Edit

Add topics

3 commits | 2 branches | 0 releases | 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

Ferdinando Santacroce Add a sentence to readme — Latest commit 41bdbe6 12 minutes ago

README.md — Add a sentence to readme — 12 minutes ago

README.md

Cookbook

This repository contains recipes I like to share with my friends. You will find traditional Italian recipes, often revamped by me.

Welcome into the 2nd edition of Git Essentials book!

Working with a public server on GitHub

Pushing a new branch to the remote

1. Create a new branch, for instance Risotti.

2. Add to it a new file, for example, Risotto-alla-Milanese.md, and commit it.

3. Push the branch to the remote using git push -u origin Risotti.

Working with a public server on GitHub

[11] ~/Cookbook (master)$ git branch Risotti[12] ~/Cookbook (master)$ git checkout Risotti[13] ~/Cookbook (Risotti)$ notepad Risotto-alla-Milanese.md[14] ~/Cookbook (Risotti)$ git add Risotto-alla-Milanese.md[15] ~/Cookbook (Risotti)$ git commit -m "Add risotto alla milanese recipe ingredients"[Risotti b62bc1f] Add risotto alla milanese recipe ingredients 1 file changed, 15 insertions(+) create mode 100644 Risotto-alla-Milanese.md[16] ~/Cookbook (Risotti)$ git push -u origin RisottiTotal 0 (delta 0), reused 0 (delta 0)Branch Risotti set up to track remote branch Risotti from origin.To https://github.com/fsantacroce/Cookbook.git * [new branch] Risotti -> Risotti

Working with a public server on GitHub

- If you want to see remotes actually configured in your repository, you can type a simple git remote command, followed by -v (--verbose) to get some more details:

```
[17] ~/Cookbook (master)$ git remote -vorigin
https://github.com/fsantacroce/Cookbook.git (fetch)origin
https://github.com/fsantacroce/Cookbook.git (push)
```

Working with a public server on GitHub

- To better understand the way our repository is now configured, try to type git remote show origin:

[18] ~/Cookbook (master)$ git remote show origin* remote origin Fetch URL: https://github.com/fsantacroce/Cookbook.git Push URL: https://github.com/fsantacroce/Cookbook.git HEAD branch: master Remote branches: Pasta tracked Risotti tracked master tracked Local branches configured for 'git pull': Risotti merges with remote Risotti master merges with remote master Local refs configured for 'git push': Risotti pushes to Risotti (up to date) master pushes to master (fast-forwardable)

Working with a public server on GitHub

Create a new local repository to publish, following these simple steps:

- Go to our local repositories folder.
- Create a new HelloWorld folder.
- In it place a new repository, as we did in first lesson.

Working with a public server on GitHub

- Add a new README.md file and commit it:

[19] ~$ mkdir HelloWorld[20] ~$ cd HelloWorld/[21] ~/HelloWorld$ git initInitialized empty Git repository in C:/Users/san/Google Drive/Packt/PortableGit/home/HelloWorld/.git/[22] ~/HelloWorld (master)$ echo "Hello World!" >> README.md[23] ~/HelloWorld (master)$ git add README.md[24] ~/HelloWorld (master)$ git config user.name "Ernesto Lee"[25] ~/HelloWorld (master)$ git config user.email "socrates73@gmail.com"[26] ~/HelloWorld (master)$ git commit -m "First commit"[master (root-commit) 5b41441] First commit 1 file changed, 1 insertion(+) create mode 100644 README.md[27] ~/HelloWorld (master)

Working with a public server on GitHub

Search GitHub | Pull requests | Issues | Marketplace | Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: fsantacroce / Repository name: HelloWorld

Great repository names are short and memorable. Need inspiration? How about stunning-octo-sniffle.

Description (optional)

A simple repository for tests

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Create repository

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

Working with a public server on GitHub

- Adding a remote to a local repository
- To publish our HelloWorld repository, we simply have to add its first remote; adding a remote is quite simple: git remote add origin <remote-repository-url>

- So, this is the full command we have to type in the Bash shell:

```
[27] ~/HelloWorld (master)$ git remote add origin
https://github.com/fsantacroce/HelloWorld.git
```

Working with a public server on GitHub

Pushing a local branch to a remote repository

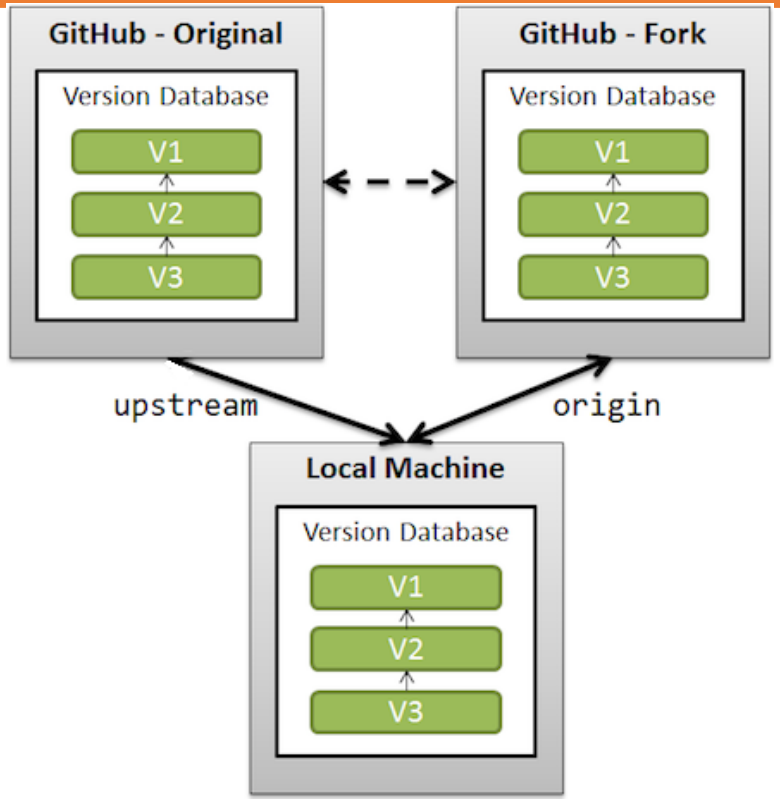
- After that, push your local changes to the remote using git push -u origin master:

[28] ~/HelloWorld (master)$ git push -u origin masterCounting objects: 3, done.Writing objects: 100% (3/3), 231 bytes | 0 bytes/s, done.Total 3 (delta 0), reused 0 (delta 0)Branch master set up to track remote branch master from origin.To https://github.com/fsantacroce/HelloWorld.git * [new branch] master -> master

Working with a public server on GitHub

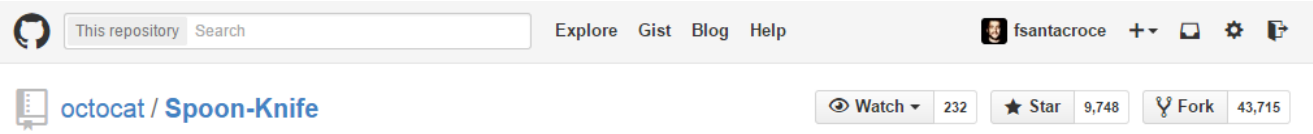
- When you fork on GitHub, what you get is essentially a server-side clone of the repository on your GitHub account; if you clone your forked repository locally, in the remote list, you will find an origin that points to your account repository, while the original repository will assume the upstream alias (you have to add it manually anyway):

Working with a public server on GitHub

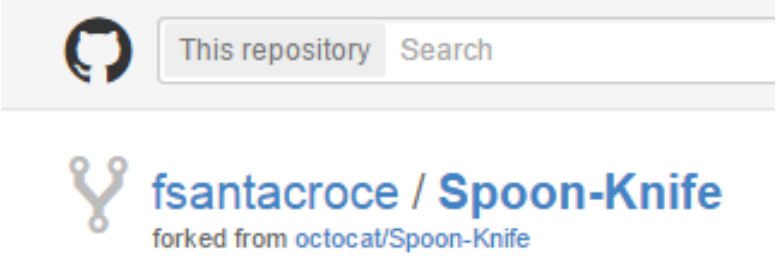


Working with a public server on GitHub

- Fork the repository using the Fork button at the right of the page:



- After a funny photocopy animation, you will get a brand-new Spoon-Knife repository in your GitHub account:



Working with a public server on GitHub

- Now, you can clone that repository locally, as we did before:

[1] ~$ git clone https://github.com/fsantacroce/Spoon-Knife.gitCloning into 'Spoon-Knife'...remote: Counting objects: 19, done.remote: Total 19 (delta 0), reused 0 (delta 0), pack-reused 19Unpacking objects: 100% (19/19), done.[2] ~$ cd Spoon-Knife[3] ~/Spoon-Knife (master)$ git remote -vorigin https://github.com/fsantacroce/Spoon-Knife.git (fetch)origin https://github.com/fsantacroce/Spoon-Knife.git (push)

Working with a public server on GitHub

- As you can see, the upstream remote is not present, you have to add it manually; to add it, use the git remote add command:

```
[4] ~/Spoon-Knife (master)$ git remote add upstream
https://github.com/octocat/Spoon-Knife.git[5] ~/Spoon-Knife
(master)$ git remote -vorigin
https://github.com/fsantacroce/Spoon-Knife.git (fetch)origin
https://github.com/fsantacroce/Spoon-Knife.git (push)upstream
https://github.com/octocat/Spoon-Knife.git (fetch)upstream
https://github.com/octocat/Spoon-Knife.git (push)
```

Working with a public server on GitHub

Creating a pull request

- To create a pull request, you have to go on your GitHub account and make it directly from your forked account; but first, you have to know that pull requests can be made only from separated branches.
-
- At this point of the course, you are probably used to creating a new branch for a new feature or refactor purpose, so this is nothing new, isn't it?

Working with a public server on GitHub

- To make an attempt, let's create a local TeaSpoon branch in our repository, commit a new file, and push it to our GitHub account:

```
[6] ~/Spoon-Knife (master)$ git branch TeaSpoon[7] ~/Spoon-Knife (master)$ git checkout TeaSpoonSwitched to branch 'TeaSpoon'[8] ~/Spoon-Knife (TeaSpoon)$ vi TeaSpoon.md[9] ~/Spoon-Knife (TeaSpoon)$ git add TeaSpoon.md[10] ~/Spoon-Knife (TeaSpoon)$ git commit -m "Add a TeaSpoon to the cutlery"[TeaSpoon 62a99c9] Add a TeaSpoon to the cutlery1 file changed, 2 insertions(+) create mode 100644 TeaSpoon.md[11] ~/Spoon-Knife (TeaSpoon)$ git push origin TeaSpoonCounting objects: 3, done.Delta compression using up to 8 threads.Compressing objects: 100% (3/3), done.Writing objects: 100% (3/3), 417 bytes | 0 bytes/s, done.Total 3 (delta 0), reused 0 (delta 0)To https://github.com/fsantacroce/Spoon-Knife.git   d0dd1f6..62a99c9  TeaSpoon -> TeaSpoon
```

Working with a public server on GitHub

This repository | Search | Pull requests | Issues | Marketplace | Explore

fsantacroce / Spoon-Knife
forked from octocat/Spoon-Knife

Unwatch 1 | Star 0 | Fork 93,201

Code | Pull requests 0 | Projects 0 | Wiki | Settings | Insights

This repo is for demonstration purposes only.

Edit

Add topics

3 commits | 4 branches | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is even with octocat:master. Pull request | Compare

octocat Pointing to the guide for forking — Latest commit d0dd1f6 on Feb 13, 2014

README.md	Pointing to the guide for forking	4 years ago
index.html	Created index page for future collaborative edits	4 years ago
styles.css	Create styles.css and updated README	4 years ago

README.md

Well hello there!

This repository is meant to provide an example for *forking* a repository on GitHub.

Creating a *fork* is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit *Pull Requests* to help make other people's projects better by offering your changes up to the original project. Forking is at the core of social coding at GitHub.

After forking this repository, you can make some changes to the project, and submit a Pull Request as practice.

For some more information on how to fork a repository, check out our guide, "Forking Projects"". Thanks! 💖

Working with a public server on GitHub

This repository Search Pull requests Issues Marketplace Explore

octocat / Spoon-Knife

Watch 310 Star 9,998 Fork 93,201

Code Issues 773 Pull requests 5,000+ Projects 0 Wiki Insights

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base fork: octocat/Spoon-Knife base: master ... head fork: fsantacroce/Spoon-Knife compare: master 1

Choose a head branch

Branch, tag, commit, or history marker

TeaSpoon 2

change-the-title

master

test-branch

test #12724
No description available

changed #9216
Here are some changes...

update the title, add index back up file #8716
update the title, add index back up file

Masterrrr #8480
No description available

View pull request

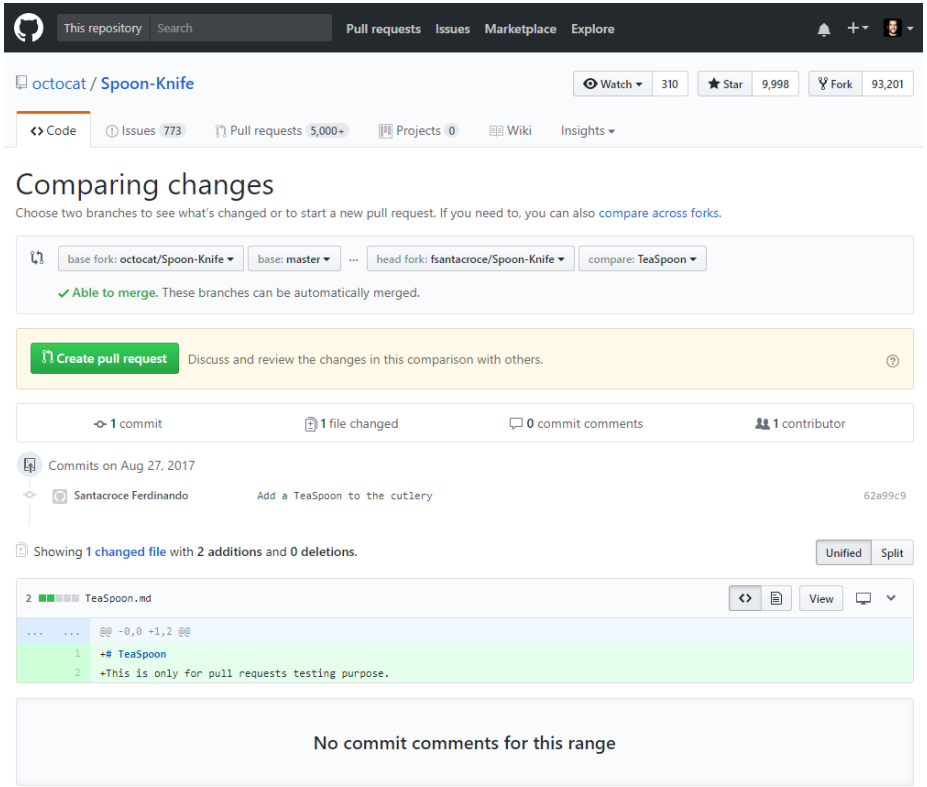
added new file #5838
fasfd

View pull request

There isn't anything to compare.
octocat:master and fsantacroce:master are identical.

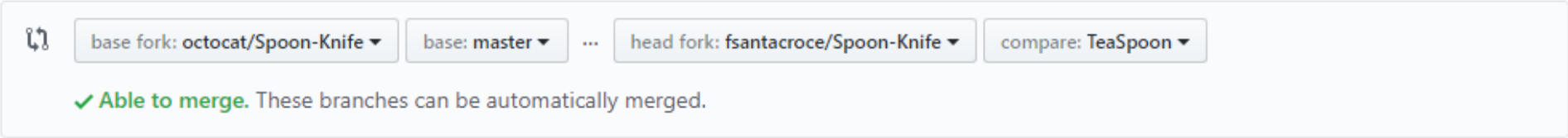
Working with a public server on GitHub

- Go to the branches combo (1), select TeaSpoon branch (2), and then GitHub will show you something similar to the following screenshot:



Working with a public server on GitHub

- In the top left corner of the preceding screenshot, you will find what branches GitHub is about to compare for you; take a look at details in the following image:



Working with a public server on GitHub

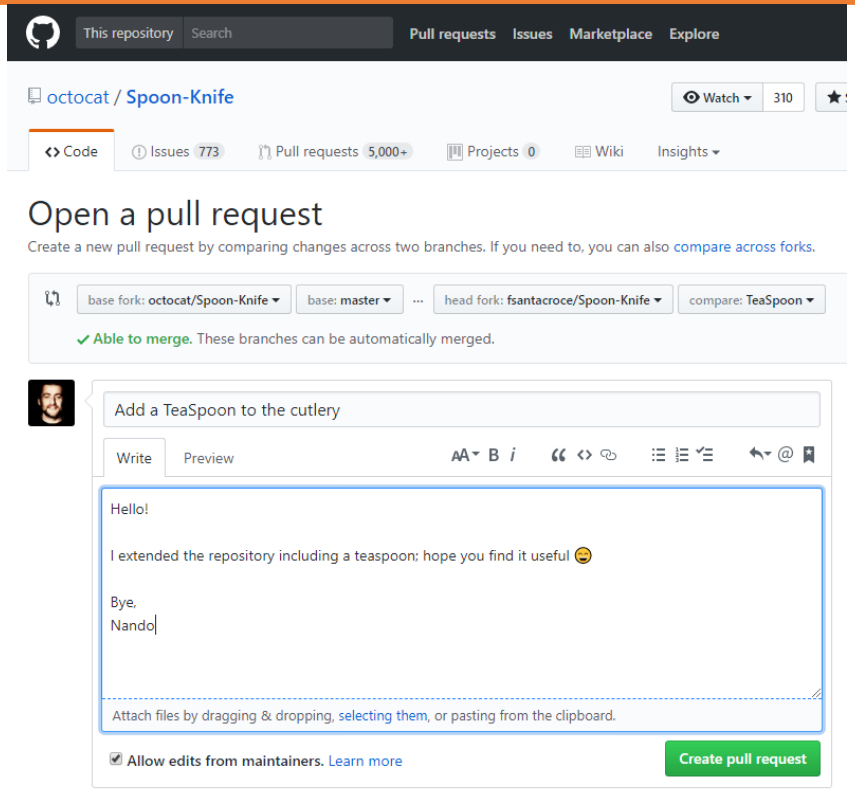
This means that you are about to compare your local TeaSpoon branch with the original master branch of the octocat user.
At the end of the page, you can see all the different details (files added, removed, changed, and so on):

2 TeaSpoon.md View

```
... ... @@ -0,0 +1,2 @@
1 +# TeaSpoon
2 +This is only for pull requests testing purpose.
```

Working with a public server on GitHub

- Now, you can click on the green Create pull request button; the window in the following screenshot will appear:



Summary

- In this lesson, we finally got in touch with the Git ability to manage multiple remote copies of repositories.
- This gives you a wide range of possibilities to better organize your collaboration workflow inside your team.
- In the next lesson, you will learn some advanced techniques using well-known and niche commands.

4. Git Fundamentals - Niche Concepts, Configurations, and Commands

Git Fundamentals - Niche Concepts, Configurations, and Commands

- This lesson is a collection of short but useful tricks to make our Git experience more comfortable.

- In the first three lessons, we learned all the concepts we need to take the first steps into versioning systems using the Git tool; now it's time to go a little bit in depth to discover some other powerful weapons in the Git arsenal, and how to use them (without shooting yourself in the foot, preferably).

Dissecting Git configuration

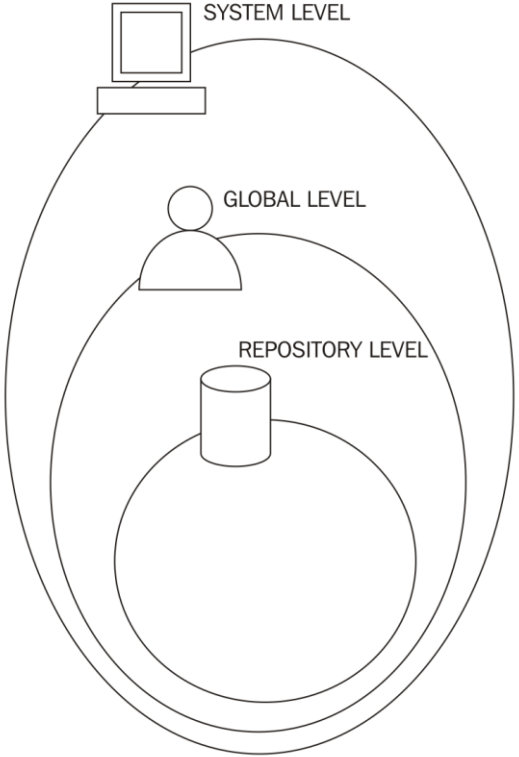
Configuration levels

In Git, we have three configuration levels:

1. System
2. Global (user-wide)
3. Repository

Dissecting Git configuration

- The following figure will help you to better understand these levels:



Dissecting Git configuration

System level

This configuration is stored in the gitconfig file usually located in:

- Windows: C:\Program Files\Git\etc\gitconfig
- Linux: /etc/gitconfig
- macOS: /usr/local/git/etc/gitconfig

Dissecting Git configuration

Global level

This configuration is stored in the .gitconfig file usually located in:

- Windows: C:\Users\<UserName>\.gitconfig
- Linux: ~/.gitconfig
- macOS: ~/.gitconfig

Dissecting Git configuration

Repository level

This configuration is stored in the config file located in the .git repository subfolder:

Windows: C:\<MyRepoFolder>\.git\config
Linux: ~/<MyRepoFolder>/.git/config
macOS: ~/<MyRepoFolder>/.git/config

Dissecting Git configuration

Listing configurations

- To get a list of all the configurations currently in use, you can run the git config --list command; if you are inside a repository, it will show all the configurations, from repository to system level.
- To filter the list, append optionally --system, --global or --local options to obtain only the desired level configurations:

Refer to the file 4_1.txt

Dissecting Git configuration

Typos autocorrection

- So, let's try to fix an annoying question about typing command: typos.
- I often find myself re-typing the same command two or more times; Git can help us with embedded autocorrection, but we first have to enable it.
- To enable it, you have to modify the help.autocorrection parameter, defining how many tenths of a second Git will wait before running the assumed command; so giving a help.autocorrect 10, Git will wait for a second:

Refer to the file 4_2.txt

Dissecting Git configuration

- You can see the section names within [] if you look in the configuration file; for example, in C:\Users\<UserName>\.gitconfig:

```
39      trustExitCode = true
40
41  [help]
42      autocorrect = 10
43  [push]
44      default = tracking
45
```

Normal text file | length : 12

Dissecting Git configuration

- There are two ways we can do this.
- First one: set Git to ask us the name of the branch we want to push every time, so a simple git push will have no effect.
- To obtain this, set push.default to nothing:

```
[1] ~/grocery-cloned (master)
$ git config --global push.default nothing
[2] ~/grocery-cloned (master)
$ git push
fatal: You didn't specify any refspecs to push, and
push.default is "nothing".
```

Dissecting Git configuration

- Another way to save yourself from this kind of mistake is to set the push.default parameter to simple, allowing Git to push only when there is a remote branch with the same name as the local one:

```
[3] ~/grocery-cloned (master)
$ git config --global push.default simple

[4] ~/grocery-cloned (master)
$ git push
Everything up-to-date
```

Dissecting Git configuration

Defining the default editor

- Some people really don't like vim, even only for writing commit messages; if you are one of them, there is good news: you can change it by setting the core.default config parameter:

```
[1] ~/grocery (master)
$ git config --global core.editor notepad
```

Git aliases

Shortcuts to common commands

- One thing you may find useful is to shorten common commands such as git checkout and so on; therefore, useful aliases can include the following:

```
[1] ~/grocery (master)
$ git config --global alias.co checkout
[2] ~/grocery (master)
$ git config --global alias.br branch
[3] ~/grocery (master)
$ git config --global alias.ci commit
[4] ~/grocery (master)
$ git config --global alias.st status
```

Git aliases

- Another common practice is to shorten a command, adding one or more options you use all the time; for example, set a git cm <commit message> command shortcut to the alias git commit -m <commit message>:

```
[5] ~/grocery (master)
$ git config --global alias.cm "commit -m"
[6] ~/grocery (master)
$ git cm "My commit message"
On branch master
nothing to commit, working tree clean
```

Git aliases

- The classic example is the git unstage alias:

```
[1] ~/grocery (master)
$ git config --global alias.unstage 'reset HEAD --’
```

- With this alias, you can remove a file from the index in a more meaningful way, compared to the equivalent git reset HEAD -- <file> syntax:

```
[2] ~/grocery (master)
$ git unstage myFile.txt
```

Git aliases

- Now behaves the same as:

```
[3] ~/grocery (master)
$ git reset HEAD -- myFile.txt
```

git undo

- Want a fast way to revert the last ongoing commit? Create a git undo alias:

```
[1] ~/grocery (master)
$ git config --global alias.undo 'reset --soft HEAD~1'
```

Git aliases

git last

- A git last alias is useful to read about your last commit:

```
[1] ~/grocery (master)
$ git config --global alias.last 'log -1 HEAD'
[2] ~/grocery (master)
$ git last

commit b25ffa60f44f6fc50e81181cab87ed3dbf3b172c
Author: Ernesto Lee <socrates73@gmail.com>
Date:   Thu Jul 27 15:12:48 2017 +0200Add an apricot
```

Git aliases

git difflast

- With the git difflast alias, you can see a diff against your last commit:

Refer to the file 4_3

Git aliases

Advanced aliases with external commands

- If you want the alias to run external shell commands, instead of a Git sub-command, you have to prefix the alias with a !:

```
$ git config --global alias.echo !echo
```

- Suppose you are annoyed by the canonical git add <file> plus git commit <file> sequence of commands, and you want to do it in a single shot; you can call the git command twice in sequence by creating this alias:

```
$ git config --global alias.cm '!git add -A && git commit -m'
```

Git aliases

Removing an alias

- Removing an alias is quite easy; you have to use the --unset option, specifying the alias to remove.
- For example, if you want to remove the cm alias, you have to run:

```
$ git config --global --unset alias.cm
```

Git aliases

Aliasing the git command itself

- I've already said I'm a bad typist; if you are too, you can alias the git command itself (using the default alias command in Bash):

```
$ alias gti='git'
```

Useful techniques

- Append a new fruit to the shopping list, then try to switch branch; Git won't allow you to do so, because with the checkout you would lose your local (not yet committed) changes to the shoppingList.txt file.
- So, type the git stash command; your changes will be set apart and removed from your current branch, letting you switch to another one (berries, in this case):

Refer to the file 4_4.txt

Useful techniques

- Let's take a look at the actual situation in our repository using the git log command:

Refer to the file 4_5.txt

Useful techniques

- Here is the complete list of commands:

Refer to the file 4_6.txt

- OK, let's see what happened using the git log command:

Refer to the file 4_7.txt

Useful techniques

Git commit amend - modify the last commit

- This trick is for people that don't double-check what they're doing.
- If you have pressed the enter key too early, there's a way to modify the last commit message or add that file you forgot, using the git commit command with the --amend option:

```
$ git commit --amend -m "New commit message"
```

Useful techniques

Git blame - tracing changes in a file

- Working on source code in a team, it is not uncommon to have the need to look at the last modifications made to a particular file to better understand how it evolved over time.
- To achieve this result, we can use the git blame <filename> command.
- Let's try it inside the Spoon-Knife repository to see changes made to the README.md file during a specific time:

Refer to the file 4_8.txt

Useful techniques

- Suppose now you found that the modification you are looking for is the one made in the d0dd1f61 commit; to see what happened there, type the git show d0dd1f61 command:

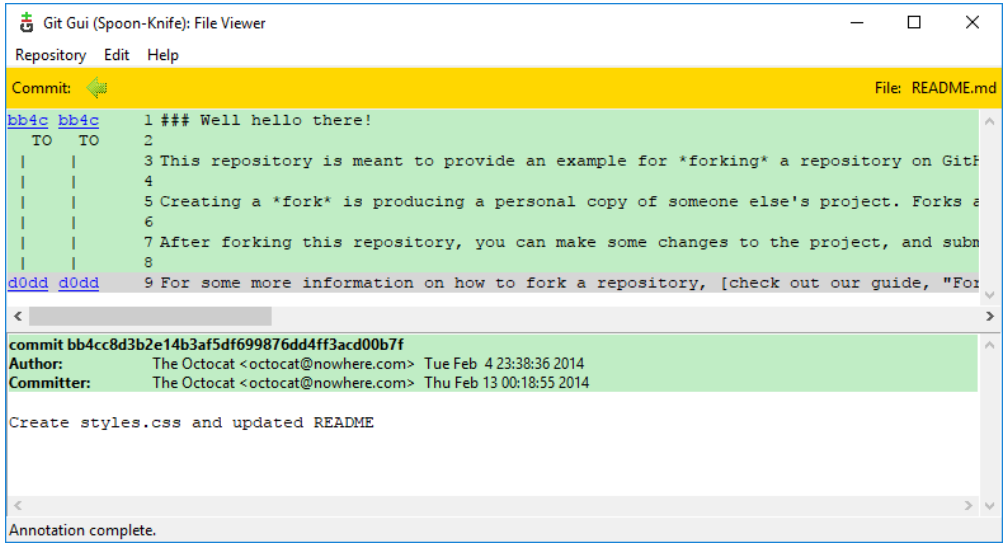
Refer to the file 4_9.txt

Useful techniques

- The last tip I want to suggest is to use the Git GUI:

[3] ~/Spoon-Knife (master)

$ git gui blame README.md



Tricks

- If you want to set up a bare repository, you only have to use the --bare option:

```
$ git init --bare NewRepository.git
```

- As you may have noticed, I called it NewRepository.git, using a .git extension; this is not mandatory, but is a common way to identify bare repositories. If you pay attention, you will note that even in GitHub every repository ends with a .git extension.

Tricks

Converting a regular repository to a bare one

- It can happen that you start working on a project in a local repository, and then you feel the need to move it to a centralized server to make it available for other people or from other locations.
- You can easily convert a regular repository to a bare one using the git clone command with the same --bare option:

```
$ git clone --bare my_project my_project.git
```

Tricks

Archiving the repository

- To archive the repository without including versioning information, you can use the git archive command; there are many output formats but the classic one is the .zip one:

```
$ git archive master --format=zip --output=../repoBackup.zip
```

Tricks

- Please note that using this command is not the same as backing up folders in a filesystem; as you will have noticed, the git archive command can produce archives in a smarter way, including only files in a branch or even in a single commit; for example, by doing this you are archiving only the last commit:

```
$ git archive HEAD --format=zip --output=../headBackup.zip
```

Tricks

Bundling the repository

- Another interesting command is the git bundle command. With git bundle, you can export a snapshot from your repository and then restore it wherever you want.
- Suppose you want to clone your repository on another computer, and the network is down or absent; with this command, you can create a repo.bundle file of the master branch:

$ git bundle create ../repo.bundle master

Tricks

- With this other command, we can restore the bundle in the other computer using the git clone command:

```
$ cd /OtherComputer/Folder
$ git clone repo.bundle repo -b master
```

Summary

- In this lesson, we enhanced our knowledge about Git and its wide set of commands. We discovered how configuration levels work, and how to set our preferences using Git by, for example, adding useful command aliases to the shell.

- Then we looked at how Git deals with stashes, providing the way to shelve then and reapply changes.