

20

MODERN WEB DEVELOPMENT TOOLS

OVERVIEW

- **Introduction to the command line**
- **CSS preprocessors (Sass)**
- **CSS postprocessors (PostCSS)**
- **“Build” tools**
- **Git version control**

Command-Line Interface

In a **command-line interface (CLI)**, you interact with the computer by typing commands directly into a **terminal** program.

The program that interprets the commands you type is called a **shell** (**bash** is the shell used on Mac and Linux).

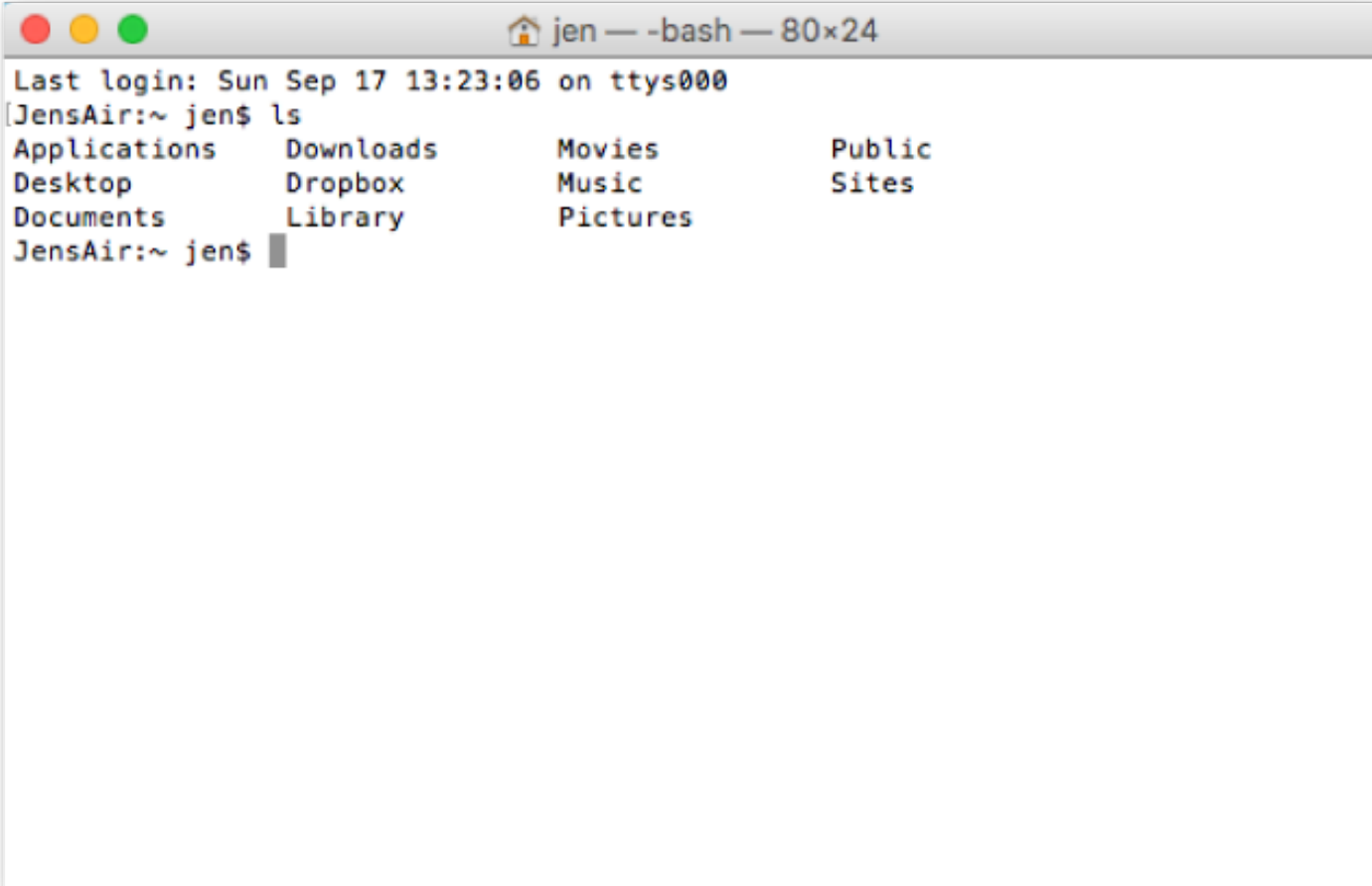
Command-line tools are popular because

- They are useful for file management on remote servers.
- It is easier to create a command-line tool than one with a full graphical user interface (GUI).

Terminal

The **Terminal** application on Mac and Linux runs the **bash shell** required for many web development tools.

Windows users can install the Cygwin bash emulator or install a full Linux environment.



```
jen — -bash — 80x24
Last login: Sun Sep 17 13:23:06 on ttys000
[JensAir:~ jen$ ls
Applications    Downloads      Movies         Public
Desktop         Dropbox       Music          Sites
Documents      Library       Pictures
JensAir:~ jen$
```

Prompts

The **command-line prompt** is a string of characters that indicates the terminal is ready to receive a command:

`$: _`

Prompts may also include the **computer name**, the **working directory**, and the **user name**:

`MyComputer:Sites jen$: _`

When you see the prompt, type your command and hit Enter.

NOTE: The **underscore** above indicates the cursor position (it may be a flashing line or rectangle on the screen).

Commands

Commands are standardized abbreviations for the task you want to perform.

Type a command after the prompt. The command is executed, and a new prompt appears.

Example: The **ls** command displays the contents of the current (working) directory:

```
JensAir:~ jen$ ls
Applications  Downloads  Movies     Public
Desktop       Dropbox    Music      Sites
Documents     Library    Pictures
JensAir:~ jen$
```

Flags

A **flag** changes how a utility operates (like an option). It follows the command name and is indicated by a single or double dash (-).

Example: Adding the **-l** (long) flag, makes the **ls** command display directory contents in a longer format that includes permission settings and creation dates:

```
JensAir:~ jen$ ls -l
total 0
drwxr-xr-x    5 jen  staff   170 Jul  8  2016 Applications
drwx-----  57 jen  staff  1938 Sep 11 09:47 Desktop
drwx-----  26 jen  staff   884 May 18 11:34 Documents
drwx-----+ 151 jen  staff  5134 Sep  3 15:47 Downloads
...
drwxr-xr-x    6 jen  staff   204 May  6  2015 Public
drwxr-xr-x   11 jen  staff   374 Jul 10  2016 Sites
JensAir:~ jen$
```

Arguments

An **argument** provides the specific information required by some functions.

Example: To change to a new directory, use the **cd** (change directory) command as well as the name of the target directory. *The directory name is the argument for the **cd** command.*

```
JensAir:~ jen$ cd Dropbox  
JensAir:Dropbox jen$_
```

To back up a level, use the "dot-dot" shorthand:

```
JensAir:Dropbox jen$ cd ..  
JensAir:~ jen$_
```


A Few More Command Examples

mv: Moves files and folders

cp: Copies files

mkdir: Creates a new empty directory

rm: Removes a file or subdirectory from the working directory permanently

man: Displays documentation (the *manual*) for a command (example: `man ls` shows manual for the `ls` command).

NOTE: For a complete list of bash commands, see ss64.com/bash.

CSS Preprocessors

CSS preprocessors allow authors to write CSS in a syntax similar to a scripting language.

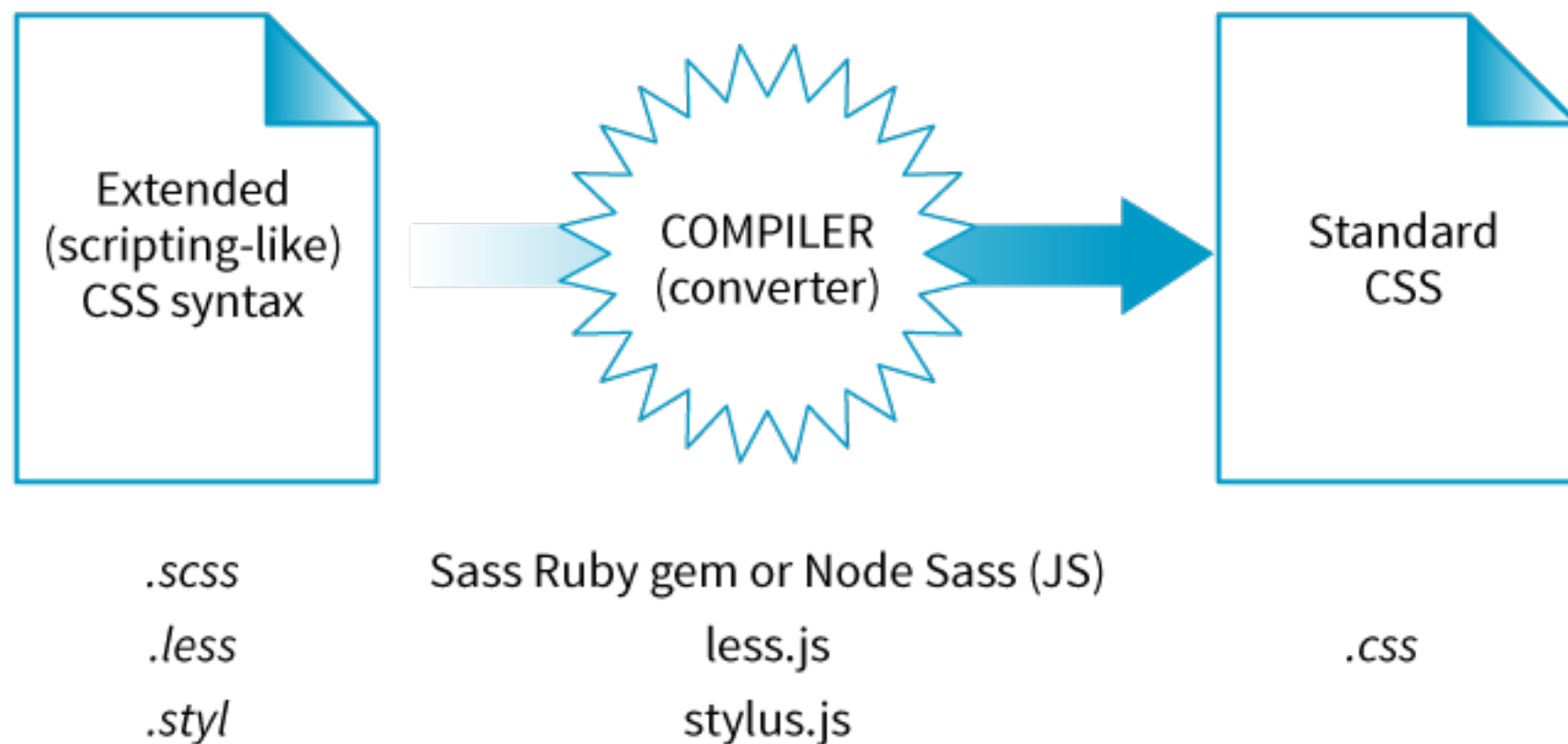
The most popular preprocessor is **Sass**, followed by **LESS** and **Stylus**.

Preprocessor syntax offers many efficiencies, including:

- Nesting styles
- Reusable variables
- Reusable sets of styles (mixins)

CSS Preprocessors (cont'd)

A **compiler** program converts the preprocessor language to a standard `.css` file the browser can read:



Nesting

(shown in Sass syntax)

Sass lets you nest style rules to match the structure of the document (it decreases the number of selectors):

Sass syntax

```
nav {  
  margin: 1em 2em;  
  ul {  
    list-style: none;  
    padding: 0;  
    margin: 0;  
    li {  
      display: block;  
      width: 6em;  
      height: 2em;  
    }  
  }  
}
```

Converted to standard CSS:

```
nav {  
  margin: 1em 2em;  
}  
nav ul {  
  list-style: none;  
  padding: 0;  
  margin: 0;  
}  
nav ul li {  
  display: block;  
  width: 6em;  
  height: 2em;  
}
```

Variables

(shown in Sass syntax)

A **variable** is a value you define once and use multiple times throughout the document.

The advantage is you can change the value in one place instead of replacing it everywhere. It also helps keep styles consistent.

Sass syntax:

```
$oreilly-red: #900;

a {
  border-color: $oreilly-red;
}
```

Converted to standard CSS:

```
a {
  border-color: #900;
}
```

Mixins

(shown in Sass syntax)

A **mixin** is set of style declarations that you can define once and reuse. It eliminates a lot of repetitive code:

Sass syntax:

```
@mixin special {  
  color: #fff;  
  background-color: #befc6d;  
  border: 1px dotted #59950c;  
}  
a.nav {  
  @include special;  
}  
a.nav: hover {  
  @include special;  
  border: 1px yellow solid;  
}
```

Converted to standard CSS:

```
a.nav {  
  color: #fff;  
  background-color: #befc6d;  
  border: 1px dotted #59950c;  
}  
a.nav: hover {  
  color: #fff;  
  background-color: #befc6d;  
  border: 1px dotted #59950c;  
  border: 1px yellow solid;  
}
```

Mixins with Arguments

(shown in Sass syntax)

Use an **argument** (a placeholder value indicated with a **\$**) in a mixin to plug different values in as needed:

Sass syntax:

```
@mixin rounded($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  border-radius: $radius;  
}  
aside {  
  @include rounded(.5em);  
  background: #f2f5d5;  
}
```

Converted to standard CSS:

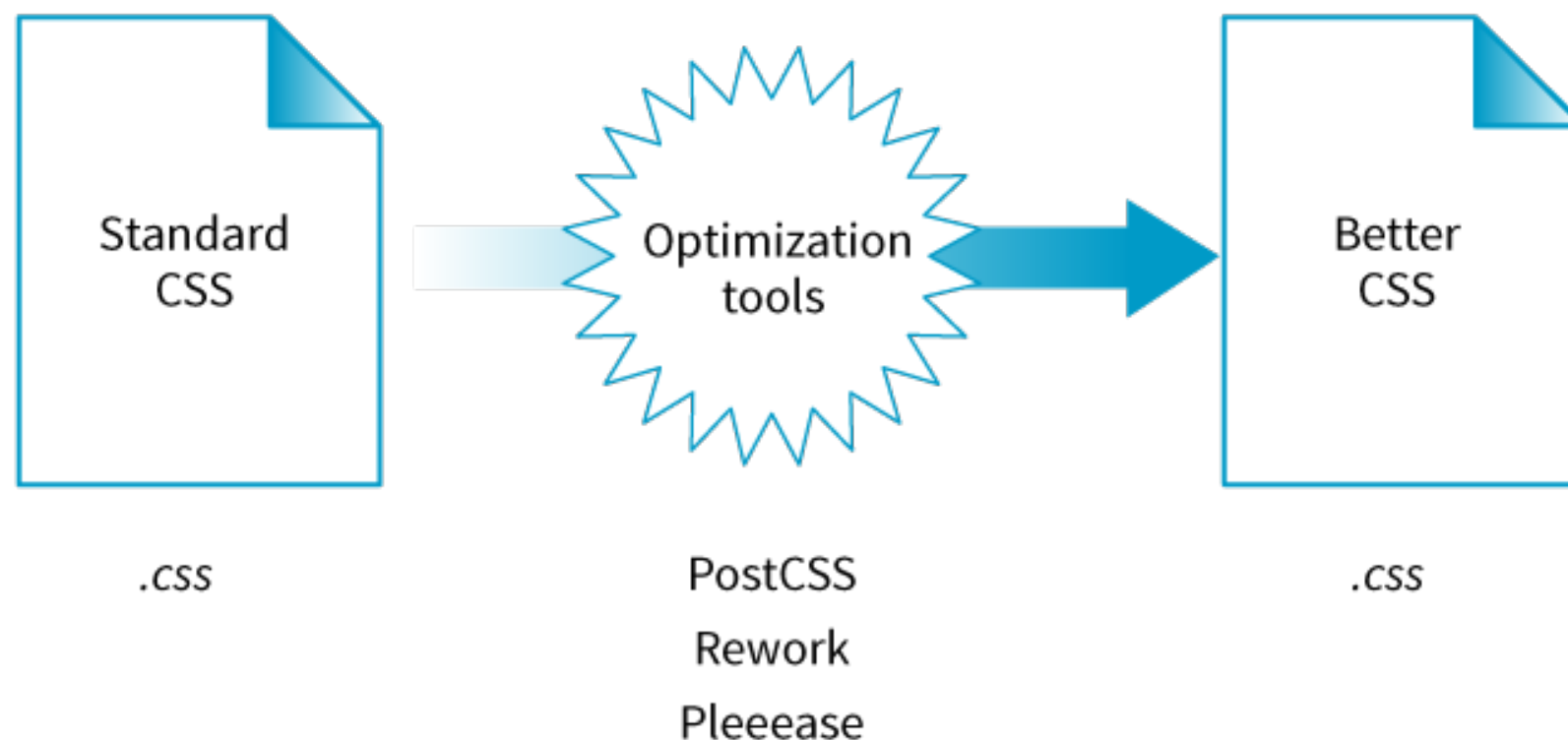
```
aside {  
  -webkit-border-radius: .5em;  
  -moz-border-radius: .5em;  
  border-radius: .5em;  
  background: #f2f5d5;  
}
```

NOTE: This is useful for vendor prefixes, as shown in the example.

Postprocessors

Postprocessors take standard CSS and optimize it.

The most popular postprocessor is **PostCSS**. This is a JavaScript-based program (a **Node.js** module) with hundreds of community-created plug-ins that solve many CSS problems.



PostCSS Examples

A few examples of tasks handled by PostCSS plug-ins:

- Adding vendor prefixes when needed (Autoprefixer)
- Checking for CSS syntax errors (Stylelint)
- Converting rem units to pixel units (Pixrem)
- Generating fallbacks for cutting-edge CSS4 features (CSSNext)
- Inserting hacks required for old versions of IE (Fixie)

Task Runners (Build Tools)

A **task runner** is a program that automates common production processes to make your workflow more efficient.

A **build tool** generates web pages from multiple components, such as templates and a database.

The most popular tools are **Grunt** and **Gulp**.

They're built on the open source **Node.js** JavaScript framework.

Common Build Tool Tasks

- Running CSS **pre- and postprocessors**
- **Concatenation:** Assembling module-based style sheets and scripts into master files for publication
- **Compression** and **minification:** Removing unnecessary whitespace and line returns to reduce file size
- Optimizing images in batches to reduce file size
- Committing and pushing changes to Git
- Building final HTML files from templates and data

Version Control with Git

A **version control system (VCS)**

- Saves versions of work that you can go back to
- Allows multiple people to work on a shared set of files

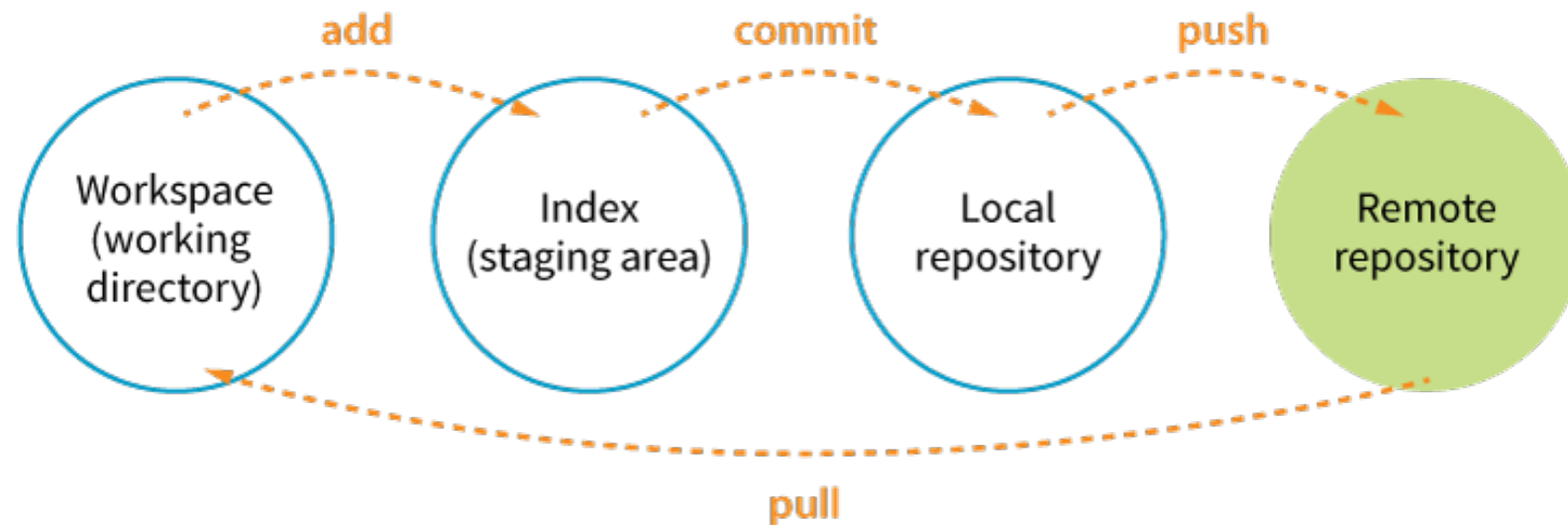
Git is the most popular VCS for web development.

GitHub is a service that provides servers for Git projects, for free or for a fee.

How Git Works

- **Git keeps a copy** of every revision of your files and folders.
- Every change (called a **commit**) is logged in with
 - A unique ID (generated by Git)
 - A message describing the change (written by the user)
 - Other metadata
- All past versions and the commit log are stored in a **repository** (also called a **repo**).
- Git is a **distributed version control system**, meaning everyone has their own copy of the repo and can work locally and offline.

Git Process and Structure



Working directory

The directory of files on your computer in which you do your actual work

Staging area

Files that have been added to Git's index but that have not yet been committed

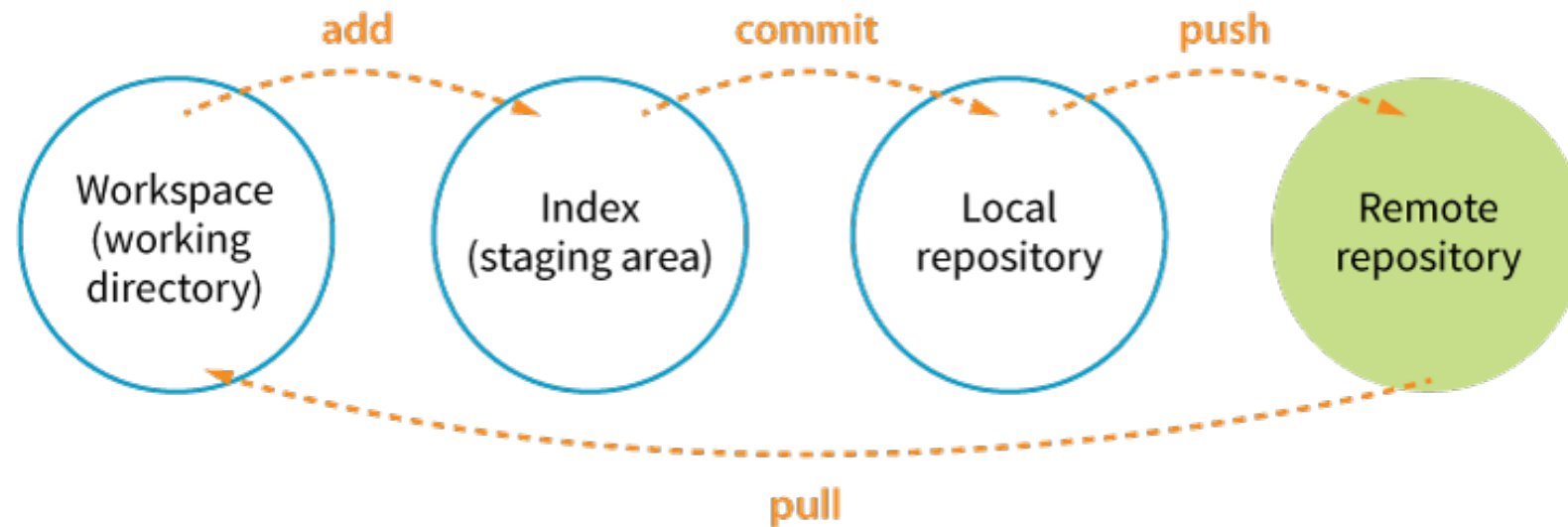
Local repository

A copy of the Git repository that resides on your own computer

Remote repository

A copy of a repository that resides on a shared server, often serving as a master copy for a team to share

Git Process and Structure (cont'd)



add (staging)

Making Git aware of a file so it can track it. When you add a file, it is included in the index but not yet committed.

commit

To log a change to the current version of the repository, usually at a logical point in the workflow

push

To move data from your local repo to a remote repo

pull

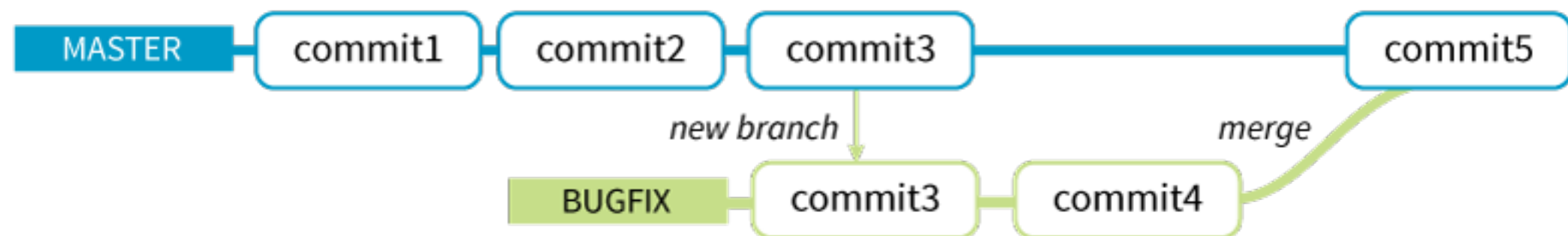
To update your local repo with data from the shared remote repo

Branching

A **branch** is a sequential series of commits. You can think of it as a thread of development.

Projects usually have a primary branch called the **master**, which is the official version of the project.

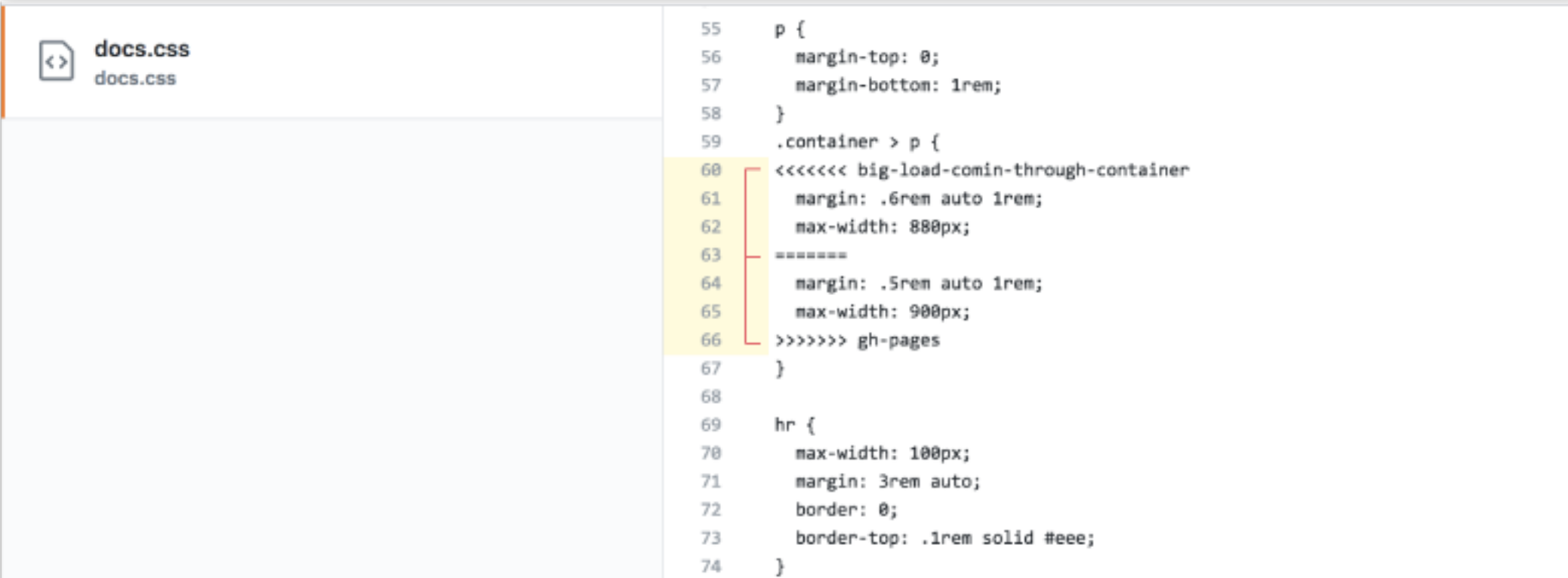
You can start a new branch to work on a particular feature or bug fix:



Merging

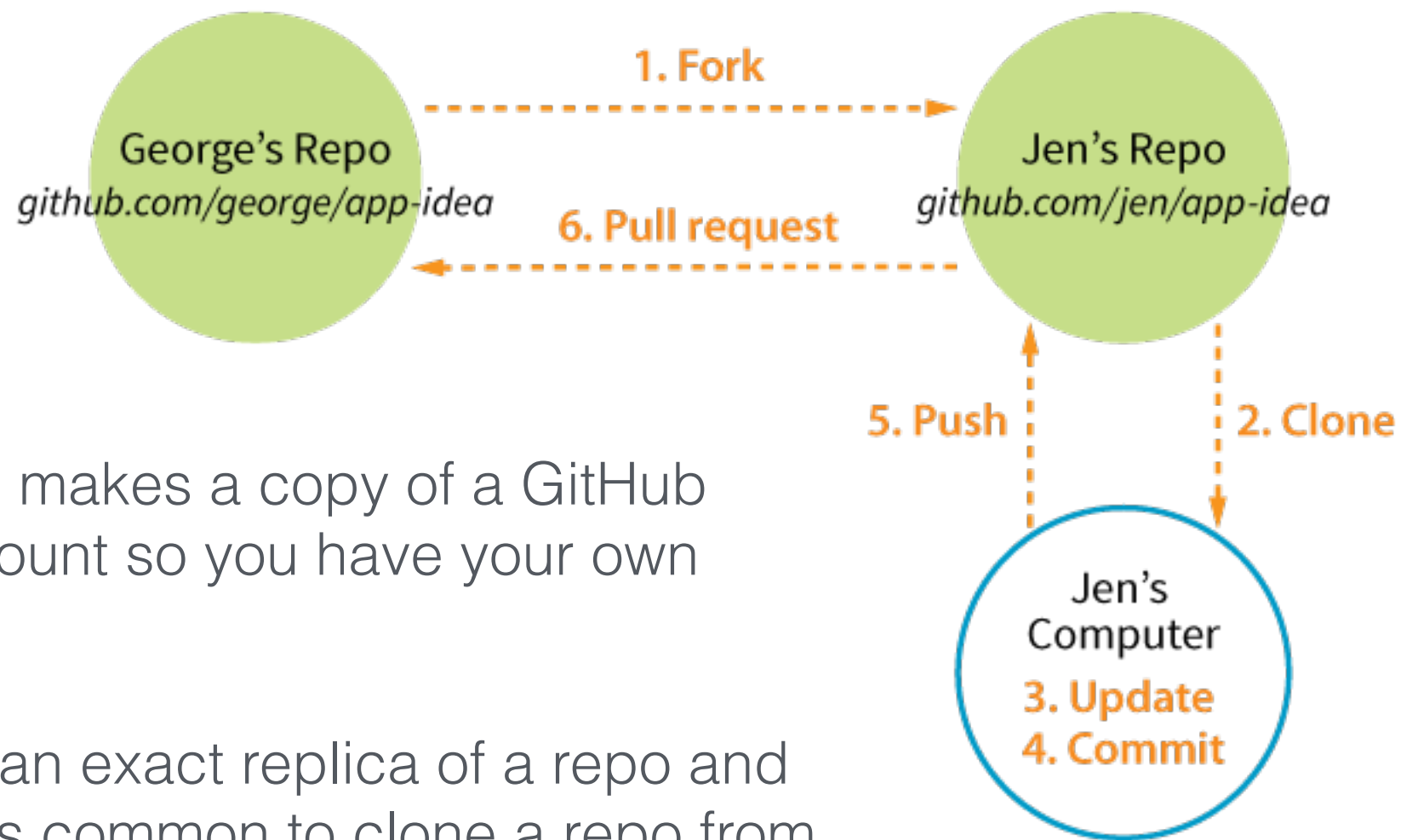
You can **merge** the commits from one branch into another (such as from a feature branch into the master).

If Git finds **conflicts** (different changes to the same line of code), it gives you a report of conflicts you need to fix manually:



```
55  p {
56    margin-top: 0;
57    margin-bottom: 1rem;
58  }
59  .container > p {
60  <<<<<<< big-load-comin-through-container
61    margin: .6rem auto 1rem;
62    max-width: 880px;
63  =====
64    margin: .5rem auto 1rem;
65    max-width: 900px;
66  >>>>>>> gh-pages
67  }
68
69  hr {
70    max-width: 100px;
71    margin: 3rem auto;
72    border: 0;
73    border-top: .1rem solid #eee;
74  }
```

Forking and Cloning



Forking (a GitHub term) makes a copy of a GitHub repo to your GitHub account so you have your own copy to play with.

Cloning means making an exact replica of a repo and everything it contains. It's common to clone a repo from a remote server to your local computer.

If you want your work to be merged into the original repo, you ask the owner to *pull* in your changes (called a **pull request**).