230517_2반 실습

컴퓨터(Computer)

컴퓨터의 이해

컴퓨터란?

- **컴퓨터**(영어: computer, 문화어: **콤퓨터**, **콤퓨타**, 순화어: 전산기(電算機), 셈틀)는 방대한 정보(데이터)를 저장하고 처리할 수 있는 전자적 기계 장치이다.
- 전자회로를 이용하여 자동적으로 계산이나 데이터를 처리하는 기계로 프로그래밍이 가능하다.

역사

1. 고대로부터 주판을 비롯해 계산을 돕는 많은 도구
2. 중세에 들어와서는 유럽과 서남아시아 지역에서 천체의 움직임을 예측하거나 낮과 밤의 길이를 계산하기 위한 정밀한 기계식 계산기가 발명되었으나, 천체 관측 외의 계산에는 사용할 수 없었다.
3. 1623년 독일의 학자 빌헬름 시카르트가 6자리 숫자의 덧셈과 뺄셈을 수행할 수 있는 최초의 기계식 계산기를 발명
4. 1642년에 당시 19세였던 블레즈 파스칼이 10진수의 덧셈과 뺄셈을 계산할 수 있는 기계식 계산기를 발명
5. 1822년 영국 수학자 찰스 배비지는 다항 함수와 로그 함수, 삼각함수 등을 계산할 수 있는 기계식 계산기인 차분기관을 설계하였으나, 당시의 기술로는 비용이 너무 비싸 이 기계의 실물은 1855년에 이르러서야 제작
6. 1937년 벨 연구소에서 일하고 있던 미국의 수학자이자 물리학자 조지 스티비츠는 이진법을 사용하는 최초의 전자식 디지털 계산기를 개발
7. 1939년에는 선형대수학의 문제를 풀기 위해 개발된 전자식 디지털 컴퓨터인 아타나소프-베리 컴퓨터가 개발되었으나 다른 용도로 프로그래밍할 수 없었으므로 다른 목적으로 사용할 수는 없었다.
8. 1946년에 미국에서 개발된 에니악(ENIAC)
9. 1945년 미국의 컴퓨터 과학자 존 폰 노이만은 프로그램을 기억장치에 내장하는 방식의 컴퓨터를 제안하였고 그의 제안을 바탕으로 EDVAC 설계

10. 1970년대 말부터 개인용 컴퓨터(PC)가 보편화

> 출처 : https://ko.wikipedia.org/wiki/컴퓨터

컴퓨터의 구조

컴퓨터의 구성

- 컴퓨터는 저장, 처리, 조작 하는 장치이기 때문에 기본적으로 CPU, 메모리, 디스크로 구성되어 있다.

CPU(Central Processing Unit : 중앙처리장치)

- 컴퓨터에서 기억, 해석, 연산, 제어라는 4대 주요 기능을 관할하는 장치

Memory(Main Memory : 주기억장치)

- CPU는 연산을 수행한 후에 메인 메모리에 데이터를 저장하거나 필요한 데이터가 요구되는데, 이 때 CPU가 직접 접근할 수 있는 기억 장치
- RAM은 Random Access Memory의 약자로, 어느 위치에 저장된 데이터든지 접근하는 데 동일한 시간이 걸리는 메모리
- 컴퓨터의 CPU가 현재 처리중인 데이터나 명령만을 일시적으로 저장하는 휘발성 메모리이다.
- 따라서 전원이 꺼지면 메인 메모리에 저장된 내용들은 모두 사라진다.
- 보조기억장치보다 접근속도가 빠르다.
- 매번 메인 메모리에 직접 접근하는 것은 비효율적이므로, CPU와 메인 메모리 속도를 맞추기 위해 캐시가 존재한다.

Storage(Secondary Storage : 보조기억장치)

- 주기억장치의 용량 부족을 보완하기 위하여 쓰는 외부 기억 장치
- 주기억장치보다 속도는 느리지만 휘발성이 없어 드라이브에 저장된 정보는 컴퓨터의 전원꺼진 후에도 유지
- HDD(Hard Disk Driver)

- SSD(Solid State Driver)

I/O Device(Input/Output Device : 입출력 장치)

컴퓨터 구조 비교

- 작업자 : CPU
- 작업 테이블 : Memory
- 창고 : Storage

반응형 웹

반응형 웹 개요

반응형 웹이란?

- 하나의 웹사이트로 데스크탑 PC, 스마트폰, 태블릿 PC 등 접속하는 디스플레이의 종류에 따라 화면의 크기가 자동으로 변하도록 만든 웹 페이지

미디어 쿼리(Media Query)

미디어 쿼리란?

- 미디어 쿼리(`Media Query`)는 CSS3 부터 지원이 되는 CSS기술로 미디어 타입, 화면 크기 등을 기준으로 다른 스타일 시트를 적용할 수 있도록 해줍니다. 이를 이용해서 화면 크기가 변할때 스타일을 바뀌도록 해서 반응형 웹을 구현할 수 있습니다.

미디어 쿼리 사용법

1. 미디어 쿼리 형식
    - 문법

```
@media (조건문) { 실행코드 }
```

    - 예제

```
@media screen and (max-width: 768px) {
  body {
    background-color: lightgreen;
  }
}
```

    - **@media (max-width: 768px) {...}** 처럼 **미디어 타입**을 생략하면 **all** 이 기본값이 되어 모든 미디어 타입에 적용이 됩니다.
    - 미디어 타입에 사용되는 값을 여러종류가 있지만 웹 사이트를 만드는데는 **screen** 을 사용하거나 **all** 을 사용하는것이 일반적입니다.
    - 미디어 유형
        - `all`
        - `print`
        - `screen`
        - `speech`
2. 모바일우선 - min-width 사용 (최소 ~ 부터 적용)
    - 작은 가로폭부터 큰 가로폭 순서로 만드는 것

    ※ 모바일 = 작다 = min, 모바일기기는 해상도가 작기 때문에 작은게 먼저 조건에 부합

```
//기본으로 작성되는 CSS는 576px보다 작은 화면에서 작동 됩니다.
// 가로모드 모바일 디바이스 (가로 해상도가 576px 보다 큰 화면에 적용)
@media (min-width: 576px) { ... }

// 태블릿 디바이스 (가로 해상도가 768px 보다 큰 화면에 적용)
@media (min-width: 768px) { ... }

// 테스크탑 (가로 해상도가 992px 보다 큰 화면에 적용)
@media (min-width: 992px) { ... }

// 큰화면 데스크탑 (가로 해상도가 1200px 보다 큰 화면에 적용)
@media (min-width: 1200px) { ... }
```

3. 데스트탑우선 - max-width 사용 (최대 ~ 까지 적용)
    - 큰 가로폭부터 작은 가로폭 순서로 만드는 것

    ※ desktop =크다 = max, 데스크탑은 해상도가 크기 때문에 큰게 먼저 조건에 부합

```
// 기본 CSS를 작성합니다.
// 기본으로 작성되는 CSS는 1199px보다 큰 화면에서 작동 됩니다.
// 세로모드 모바일 디바이스 (가로 해상도가 576px 보다 작은 화면에 적용)
@media (max-width: 575px) { ... }

// 가로모드 모바일 디바이스 (가로 해상도가 768px 보다 작은 화면에 적용)
@media (max-width: 767px) { ... }

// 태블릿 디바이스 (가로 해상도가 992px 보다 작은 화면에 적용)
@media (max-width: 991px) { ... }

// 테스크탑 (가로 해상도가 1200px 보다 작은 화면에 적용)
@media (max-width: 1199px) { ... }
```

4. Break Point

```
/* 가로 해상도가 576px 보다 작은 화면에 적용) */
@media only screen and (max-width: 576px) {...}

/* 가로 해상도가 576px 보다 작은 큰화면 적용) */
@media only screen and (min-width: 576px) {...}

/* 가로 해상도가 768px 보다 작은 큰화면에 적용) */
@media only screen and (min-width: 768px) {...}

/* 가로 해상도가 992px 보다 작은 큰화면에 적용) */
@media only screen and (min-width: 992px) {...}

/* 가로 해상도가 1200px 보다 작은 큰화면에 적용) */
@media only screen and (min-width: 1200px) {...}
```

※ 브레이크 포인트는 미세하게 차이가 나지만, 크게 의미있진 않다.

```
/* 세로모드 모바일 디바이스 (가로 해상도가 576px 보다 작은 화면에 적용) */
@media (max-width: 575px) {...}

/* 가로모드 모바일 디바이스 (가로 해상도가 576px보다 크고 768px 보다 작은 화면에 적용)  */
@media (min-width: 576px) and (max-width: 767px) {...}

/*태블릿 디바이스 (가로 해상도가 768px보다 크고 991px 보다 작은 화면에 적용)  */
@media (min-width: 768px) and (max-width: 991px) {...}

/* 데스크탑 (가로 해상도가 992px보다 크고 1199px 보다 작은 화면에 적용)  */
```

```
@media (min-width: 992px) and (max-width: 1199px) {...}

/* 큰화면 데스크탑 (가로 해상도가 1200px 보다 큰 화면에 적용)  */
@media (min-width: 1200px) {...}
```

- 사용예

```
#media-320, #media-768, #media-1024, #media-1025 {
    display: none;
    height: 0px;
    overflow: hidden;
}

@media all and (max-width: 320px) {
    #media-320 { display: block; }
}
@media all and (min-width: 321px) and (max-width: 768px) {
    #media-768 { display: block; }
}
@media all and (min-width: 769px) and (max-width: 1024px) {
    #media-1024 { display: block; }
}
@media all and (min-width: 1025px) {
    #media-1025 { display: block; }
}
```

```
<div id="media-320"></div>
<div id="media-768"></div>
<div id="media-1024"></div>
<div id="media-1025"></div>
```

뷰포트(viewport)

뷰포트란?

- 뷰포트(`viewport`)는 웹페이지가 사용자에게 보여지는 영역을 말합니다. 데스크탑 PC에서 브라우저의 크기를 줄이면 웹페이지의 내용이 다보여지지 않고 스크롤 해서 봐야 되는 경우가 있습니다. 이때도 브라우저에 보여지는 부분이 뷰포트입니다. 데스크탑 PC는 브라우저의 크기를 바꿔서 뷰포트의 크기를 바꿀 수 있는 것입니다.
- 반면에 휴대폰이나 태블릿의 경우는 브라우저의 크기를 변경할 수 가 없습니다.(요즘은 멀티뷰를 지원하는 기기도 있지만, 일반적이지는 않습니다.) 하지만, 워낙 다양한 기기들이 존재하기 때문에, 뷰포트의 크기도 다양합니다.

- 휴대폰이나 태블릿이 나오기 이전의 웹페이지는 데스크탑 PC용으로만 만들었기 때문에 고정된 크기를 가지는 것이 대부분이 었습니다. 이렇게 고정된 크기를 가진 웹페이지를 화면이 작은 휴대폰에서 보면 한 화면에 다보이지 않게 됩니다.

<meta>

1. `<meta>` tag 개요
    - HTML5에서 소개된 뷰포트 <meta> 태그를 사용하면 모바일 기기에서 실제 렌더링되는 영역과 뷰포트의 크기를 조절할 수 있습니다. 또한 줌 레벨도 조정할 수 있습니다. 아래는 가장 일반적으로 사용되는 설정 입니다.
    - HTML 문서에 대한 메타 데이터를 정의합니다.
    - 메타 데이터는 데이터에 대한 데이터 (정보)입니다.
    - 항상 <head> 요소 안에 들어갑니다.
    - 일반적으로 문자인코딩, 페이지 설명, 키워드, 문서 작성자 및 뷰포트 설정을 지정하는 데 사용됩니다.
    - 메타 데이터는 페이지에 표시되지 않지만 컴퓨터 구문 분석이 가능합니다.
    - 메타 데이터는 브라우저 (콘텐츠를 표시하거나 페이지를 다시로드하는 방법), 검색 엔진 (키워드) 및 기타 웹 서비스에서 사용됩니다.
2. <meta> 속성

속성	값	설명
charset	UTF-8EUC-KR등등각국의 언어 셋	HTML문서의 문자 인코딩을 지정
content	연관된 값	http-equiv 또는 이름 속성과 연관된 값을 지정합니다.
http-equiv	content-security-policycontent-typedefault-stylerefresh	콘텐츠 속성의 정보 / 값에 대한 HTTP 헤더를 제공합니다.
name	application-nameauthordescriptiongeneratorkeywordsviewport	메타 데이터의 이름을 지정합니다.

    - <meta>의 http-equiv속성
        - 위의 표에도 보다시피 콘텐츠 속성의 정보 / 값에 대한 HTTP 헤더를 제공합니다.

    - 가장 많이 사용되는 것은 익스플로러 브라우저에 호환성 보기 모드가 존재하는데, 사용자가 지원하는 브라우저에 따라 오래된 브라우저에서는 정상적으로 출력되지 않는 경우가 발생할 수 있습니다.
    - 오래된 브라우저 지원을 위해 meta태그를 작성하는 경우가 많습니다.
    - IE9에서 호환성보기 무시하기 - 특정 브라우저 지정

```
<meta http-equiv="X-UA-Compatible" content="IE=9">
```

    - 모든 익스플로러에서 호환성보기 무시

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
```

    - chrome을 사용하는 유저에게 렌더링해줌 - 가장 많이 사용하는 방식

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge; chrome=1">
```

- **width=device-width** : 페이지의 너비를 기기의 스크린 너비로 설정합니다. 즉, 렌더링 영역을 기기의 뷰포트의 크기와 같게 만들어 줍니다.
- **initial-scale=1.0** : 처음 페이지 로딩시 확대/축소가 되지 않은 원래 크기를 사용하도록 합니다. 0~10 사이의 값을 가집니다.

이것외에도 다음과 같은 값을 지정할 수 있습니다.

- **minimum-scale** : 줄일 수 있는 최소 크기를 지정합니다. 0~10 사이의 값을 가집니다.
- **maximum-scale** : 늘릴 수 있는 최대 크기를 지정합니다. 0~10 사이의 값을 가집니다.
- **user-scalable** : yes 또는 no 값을 가지며 사용자가 화면을 확대/축소 할 수 있는지는 지정합니다.

예제

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- width=device-width부분은 장치의 화면 너비를 따르도록 페이지 너비를 설정합니다(장치에 따라 다름).

- initial-scale=1.0부분은 브라우저에서 페이지를 처음로드 할 때 초기 확대 / 축소 수준을 설정합니다.
- 과거에 사용되던 리사이즈가 안되게 막던 부분들은 모바일브라우저들이 처리가 안되도록 막았으므로 작성하지 않으셔도 됩니다.

  ☞ minimum-scale=1.0은 최소사이즈를 1.0으로 처리해서 축소를 못하게 막습니다.

  ☞ maximum-scale=1.0은 최대사이즈를 1.0으로 처리해서 확대를 못하게 막습니다.

  ☞ user-scalable=no는 사용자크기변화를 no로 처리해서 크기변화를 못하게 막습니다.

  위의 줌 레벨은 1이 원래크기이고, 0.5 라면 50% 축소를 뜻합니다.

CSS

transition 속성

transition이란?

- 특정 조건 하에서 애니메이션이 동작되는 과정을 보여주고자 할 때 사용
- 전환이 일어날 요소들을 작성해주는 것!
- 속성 설정

속성	정리
transition	선택자가 변화되는 것을 시간의 흐름을 줘서 변화시키는 속성아래 속성들을 한번에 처리하는 속기법
transition-delay	변화되는 시간을 지연시키는 속성transition-delay: 1s; => 1초지연
transition-duration	변화되는 시간을 작성하는 속성transition-duration: 1s; => 초단위
transtion-property	변화되는 CSS를 구분하여 따로 처리
transition-timing-function	변화되는 시간에 ease(가속감속)처리

transition 사용하기

- `transition-property` : 효과를 적용하고자 하는 css 프로퍼티를 지정. 즉, 과정을 보고 싶은 속성을 지정
  - 기본값 : all

    - 복수의 프로퍼티를 지정하는 경우 쉼표(,)로 구분한다.
        - transition-property: width height;
    - 전환 수치를 다르게 하려면?
        - transition: width 1s, height 2s
- `transition-duration` : 효과가 나타나는데 걸리는 시간
    - 기본값 : 0s
- `transition-timing-function` : 트랜지션 효과의 속도
    - 기본값 : ease, 느리게 시작하여 점점 빨라졌다 느려지면서 종료
    - linear : 시작부터 종료까지 일정하게
    - ease-in : 느리게 시작한 후 일정한 속도에 다다르면 그 상태로 서서히
    - ease-out : 빠르게 시작해서 점점 느려짐
    - ease-in-out : 느리게 시작하여 빨라지다가 느려지면서 종료
    - 어떤 효과를 선택하든 duration 시간에 영향을 끼치지 않는다.
- `transition-delay` : 특정 조건 하에서 효과 발동
    - 기본값 : 0s
    - 초 단위(s) 또는 밀리 초 단위(ms)로 지정
- 예제

```
transition: property duration function delay

transition-property: font-size;
transition-duration: 4s;
transition-timing-function: ease-out;
transition-delay: 3s;
```

    - transition의 속성 값들을 한 줄로 작성할 때, 나머지 속성 값의 순서는 상관없지만 항상 duration이 먼저, delay가 나중에 작성되어야 한다.

예제

1. transition-duration
    - 변화가 일어나는 시간을 지정하는 속성
    - 단위는 s(초)

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="UTF-8">
        <title>CSS3 - 변화, 변형, 애니메이션</title>
        <style>
            .total{
                width: 500px; padding: 20px;
                border: 5px solid black;
            }

            .total > div{
                /* out상태에서 가로폭이 10px */
                width: 10px; height: 50px;
                margin-bottom: 10px;
                background-color: coral;

                transition-duration: 0.5s; /* css변화시 0.5초 걸려서 변화 */
            }
            .total:hover > div{
                /* .total에 마우스를 오버하면 자손인 div의 폭이 500px로 증가 */
                width: 500px;
            }
        </style>
    </head>
    <body>
        <div class="total">
            <div class="box01"></div>
            <div class="box02"></div>
            <div class="box03"></div>
            <div class="box04"></div>
            <div class="box05"></div>
        </div>
    </body>
</html>
```

- total이 안에 자손으로 5개의 요소를 감싸고 있음
- 자손은 box01~box05로 클래스를 배정
- 원래 자손은 요소들은 가로 크기를 10px
- 부모인 total에 마우스를 올리면 자손 요소들이 500px로 넓어지게 처리
- 자손 요소들에 transition-duration: 0.5s; 을 줘서 가로 크기가 변화는 것에 시간차를 둠
- 마우스를 올리면 자손들이 자연스럽게 커지는 것을 확인할 수 있다.
- 500px되는 시점이 0.5초

2. transition-delay속성

- transition효과를 지연시키는 속성
- 단위는 s(초)

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>transition(변화속성)</title>
        <style>
            .total{
                width: 500px; padding: 20px;
                border: 5px solid black;
            }

            .total > div{
                width: 10px; height: 50px;
                margin-bottom: 10px;
                background-color: coral;


                transition-duration: 0.5s;
            }
            .total:hover > div{
                width: 500px;
            }
            .total > .box01{ transition-delay: 0s; }
            .total > .box02{ transition-delay: 0.2s; }
            .total > .box03{ transition-delay: 0.4s; }
            .total > .box04{ transition-delay: 0.6s; }
            .total > .box05{ transition-delay: 0.8s; }
        </style>
    </head>
    <body>
        <div class="total">
            <div class="box01"></div>
            <div class="box02"></div>
            <div class="box03"></div>
            <div class="box04"></div>
            <div class="box05"></div>
        </div>
    </body>
</html>
```

- 각각의 자손 박스에 transition-delay시간을 각각 처리하여 변화를 지연시킴
- delay를 준 시간만큼 순차적으로 변화되는 것을 확인할 수 있다.

3. transition-property
    - 변화의 시간차를 둘 속성을 정하는 속성
    - 속성명은 여러 개를 지정해도 된다.

```
transition-property: 속성명, 속성명;
```

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>transition(변화속성)</title>
        <style>
            .total{
                width: 500px; padding: 20px;
                border: 5px solid black;
            }

            .total > div{
                width: 10px; height: 50px;
                margin-bottom: 10px;
                background-color: coral;

                transition-property: width, background-color;
                transition-duration: 1s, 0.3s;
            }
            .total:hover > div{
                width: 500px; /* 크기도 변화 */
                background-color: lightblue; /* 배경색도 변화 */
            }
        </style>
    </head>
    <body>
        <div class="total">
            <div class="box01"></div>
            <div class="box02"></div>
            <div class="box03"></div>
            <div class="box04"></div>
            <div class="box05"></div>
        </div>
    </body>
</html>
```

- 위의 코드는 total이라는 전체 박스에 마우스를 올리면 자손요소가 크기도 width: 500px;
- 배경색도 background-color: lightblue;로 변경되도록 설정했다.
- 색상을 빠르게 변하고, 가로 폭은 천천히 변경되는 것을 확인할 수 있다.

4.

상태

:hover

- 사용자의 마우스 커서가 링크 위에 올라가 있는 상태
- `:hover`는 `<h1>` 태그 등 링크가 없는 일반적인 태그나 속성에도 대부분 사용할 수 있습니다.
- 예제

```
a {
      background-color: powderblue;
      transition: background-color .5s;
}
a:hover {
      color:red;
      background-color: gold;
}

h1:hover {
      background-color:yellow;
}
```

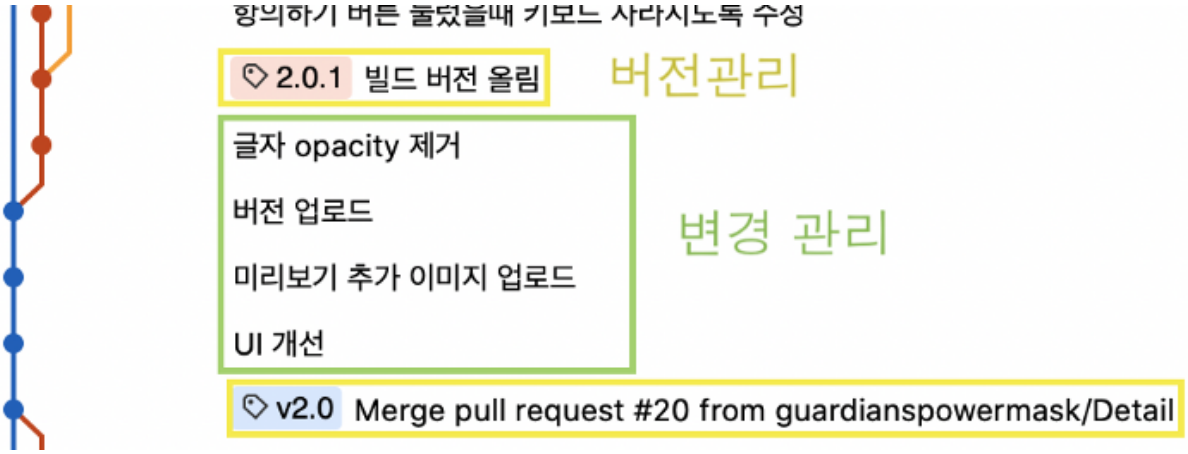
```
<html>
  <head>
    <style>
      a {
            background-color: powderblue;
            transition: background-color .5s;
      }
      a:hover {
            color:red;
            background-color: gold;
      }

      h1:hover {
            background-color:yellow;
      }
    </style>
  </head>
  <body>
    <a href="https://www.wikipedia.org">위키백과</a>
    <h1>핫!</h1>
    <a href="#">이 링크를 가리켜보세요.</a>
  </body>
</html>
```

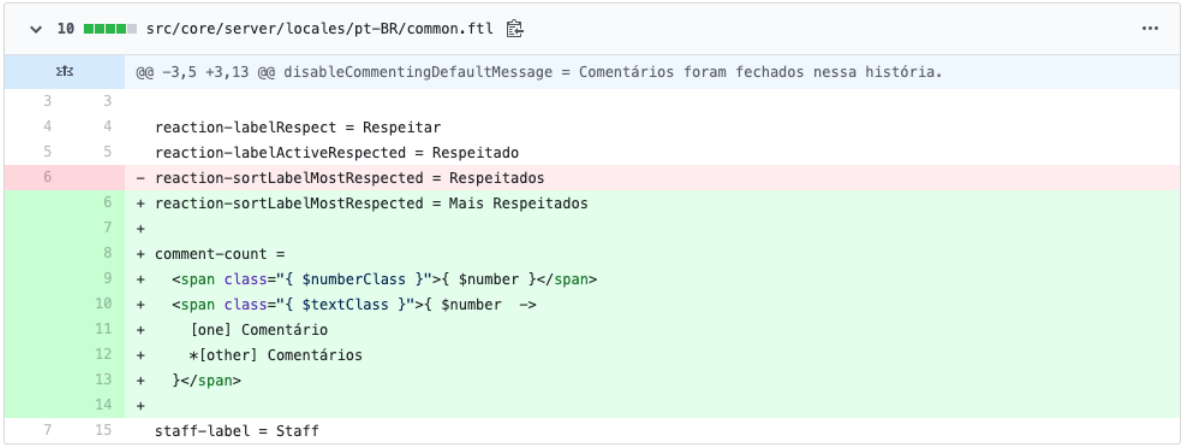
형상관리(*Software Configuration Management*)

형상관리 개요

형상관리/버전관리/변경관리

1. **변경** 관리 — **소스코드** 변경 사항에 대한 관리
2. **버전** 관리 — 변경사항을 '**버전**'이란 개념을 통해 관리.
3. **형상** 관리 — 위의 개념을 포함해 **프로젝트와 관련된 모든 변경사항**을 관리.





형상관리란?

- 소프트웨어의 **변경사항을 체계적으로 추적하고 통제하는 것**으로, 형상 관리는 일반적인 단순 버전관리 기반의 소프트웨어 운용을 좀 더 포괄적인 학술 분야의 형태로 넓히는 근간을 이야기한다.
- 형상관리는 변경사항을 체계적으로 추적, 통제한다는 것. 이 말은 어떤 문서나 파일이 변경 되었을 경우 변경된 내역을 기록하였다가 나중에 이를 찾아보아야 할 경우, 변경 원인과 변경 사항을 확인해야 할 경우에 대한 관리를 말한다.

- 많이 쓰이는 곳은 소프트웨어 개발에서 많이 쓰이지만 꼭 이에 대해서만 쓰는 것은 아니다. 예를 들어 회사 내에서 정책 문서가 있을 경우 이에 대한 변화와 왜 변경되었는지를 기록, 추후에 동일한 변경이 필요한 경우 이에 대한 과거 변경 요인들을 확인하기 위해서도 사용한다.문서의 표지 다음, 목차보다 먼저 등장하여 변경사항을 기록하도록 하는 페이지를 많이 보았을 것이다.
- 소프트웨어 개발에서 많이 사용하게 된 것을 혼자 개발하는 경우에는 문서 변경과 같은 이력 조회로써 사용할 수 있지만 여러 사람이 함께 개발하는 경우 이에 대한 내역 확인이 필수이다. 버전을 확인하여 변경사항을 확인 하고 이에 대해서 반영 및 수정하는 과정이 발생한다.또한 혼자 개발하는 경우에는 버전의 충돌이 발생하지 않지만 똑같은 파일을 다른 사람과 공유하여 개발하고 있을 경우 이에 대한 충돌 해결로도 많이 사용된다.
- *소프트웨어 구성 관리란 소프트웨어* ***소스 코드 뿐 아니라 개발 환경, 빌드 구조 등 전반적인 환경 전반적인 내역****에 대한 관리 체계를 정의하고 있다.*

형상관리 도구

1. CVS(Concurrent Version System)

- 1980년대에 만들어진 형상관리 도구로서 가장 오랫동안 사용되었으며 안정적이지만 파일관리 중 롤백이 되지 않거나 아스키코드를 지원하며 유니코드는 제한적으로 지원하고 속도가 상대적으로 느린 단점을 가지고 있다.

1. SVN(Subversion)

- CVS의 단점을 보완하기위해 2000년에 만들어진 형상관리 도구로 중앙관리만을 지원하는 특징을 가지고 있다. change set을 커밋단위로 하여 다른 사용자의 커밋과 엉키지 않고 롤백 기능을 지원한다. 처리속도가 CVS에 비해 상대적으로 빠르다. 하지만 잦은 커밋은 리비전 번호가 크게 증가할 수 있으며 개별 이력을 관리할 수 없는 단점을 가지고 있다.

1. `GIT`

- 2005년 개발된 형상관리 도구로 매우 빠른 속도를 가지고 있으며 분산형 관리 시스템을 가지고 있다. 다른 형상관리 도구에 비해 다양한 기능을 지원하며 최근 가장 대중화되어 사용되고 있다. 처음 사용시 다른 도구에 비해 사용법을 숙지하는데 어려움을 가질 수 잇으나 로컬 관리와 중앙 관리가 모두 가능하여 장소에 구애받지 않고 협업을 가능케 한다.

Git

Git이란?

- Git/Github
  - Git : 버전 컨트롤 시스템(version control system).
  - Github : 원격 저장소(remote repository).
- 현재 깃허브에서 master라는 이름 대신 main 이름을 default로 해놓은 상태지만 사용자가 master로 수정하여 사용할 수 있음.
- Git은 소프트웨어를 개발하는 기업의 핵심 자산인 소스코드를 효과적으로 관리할 수 있게 해주는 **무료**, **공개소프트웨어**.

Git의 장점

- 소스코드를 주고 받을 필요 없이, 같은 파일을 **여러 명이 동시에 작업하는 병렬 개발이 가능**하다
- 즉 브랜치를 통해 개발한 뒤, 본 프로그램에 합치는 방식(Merge)으로 개발을 진행할 수 있다.
- 분산 버전관리이기 때문에 인터넷이 연결되지 않은 곳에서도 개발을 진행할 수 있으며, 중앙 저장소가 날라가버려도 다시 원상복구할 수 있습니다.
- 팀 프로젝트가 아닌, 개인 프로젝트일지라도 GIT을 통해 버전 관리를 하면 체계적인 개발이 가능해지고, 프로그램이나 패치를 배포하는 과정도 간단해집니다. (pull을 통한 업데이트, patch 파일 배포)

Git GUI

- 너무 많은 git 명령어를 자유자재로 외울 자신이 없을 땐 GUI를 사용할 수도 있다.

ex) GitHub Desktop, SourceTree, GitKraken

Git 따라하기

- https://backlog.com/git-tutorial/kr/
- https://subicura.com/git/
- https://rogerdudler.github.io/git-guide/index.ko.html

Git 용어

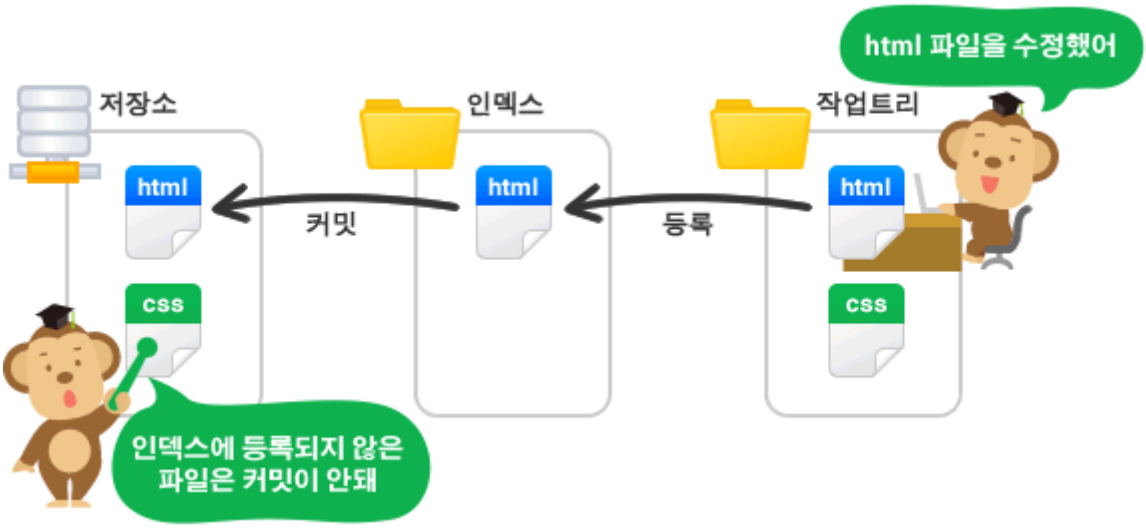
- **git init :** 버전관리 하고싶은 폴더에서 초기화를 하는 준비
- **git branch**
  - 독립적인 공간을 만든다.

    - 새로 만든 branch lab1은 master와 완전히 동일한 상태를 가진 공간.
    - 브랜치에서 수정을 한 후 커밋하면 lab1에만 기록되며 master 브랜치에는 어떤 영향도 주지 않는다.
    - 원하는 만큼 빠르게 branch를 만들 수 있다.
    - 실험 중 다른 브랜치로 돌아가야 할 때 : checkout master 로 head를 옮겨야 한다.

  (cf > 작업 중인 위치를 가르키는 가상의 커서가 존재하는데 이를 git에서는 HEAD라 한다.)

    - 실험 성공 : lab1 브랜치의 내용을 마스터 브랜치와 병합(Merge) 한다.
    - 실험 실패 : lab1 브랜치를 삭제한다.
- **checkout**
    - 독립된 작업 공간인 브랜치를 자유롭게 이동할 수 있다.
- **git commit**
    - 의미있는 수정 작업이 끝났을 때 마침을 알리는 작업
- **pull**
    - 리모트 저장소의 변경된 내용을 로컬(내 컴퓨터) 저장소에 적용하는 작업을 pull이라 한다.
- **master**
    - git init을 했을 때, default로 만들어지는 가지가 'master'이다.
- **동료와 함께 작업하려면? -> github! or bitbucket**
    - git은 'remote 저장소'를 지원한다.
    - github이 바로 remote저장소이다.
    - bitbucket은 5명까지 공동작업 가능하다. (그 이상은 유료)
    - github은 private가 유료이다.
- 작업 디렉토리(Working directory)
    - 개인 코드 작성
- Work tree
    - 폴더
- 인덱스(Index)

- 저장소와 작업 트리 사이에 존재하는 공간



- HEAD
  - 해당 브랜치의 마지막 commit

Git 명령어

1. `git init`
   - 새로운 git 저장소가 만들어집니다.
2. `git clone`
   - 저장소 받아오기
   - 로컬 저장소를 복제(clone)

```
git clone /로컬/저장소/경로
```

   - 원격 서버의 저장소를 복제

```
git clone 사용자명@호스트:/원격/저장소/경로
```

3. `git add`
   - 인덱스에 추가

```
git add <파일 이름>
```

- . : 현재 디렉토리에 있는 업데이트 된 파일을 전부 스테이징 영역으로 추가합니다.
- -A : 수정된 파일 전부를 스테이징 영역에 추가합니다.

4. `git status`
   - 현재 add 내역을 확인

```
git status
```

5. `git commit`
   - 변경 내용을 확정
   - 변경된 파일이 HEAD에 반영

```
git commit -m "이번 확정본에 대한 설명"
```

6. `git push`
   - 변경 내용 발행(push)하기
   - 변경 내용을 원격 서버로 업로드

```
git push origin master (원격 저장소 github URL)
```

7. `git reset`
   - 특정 브랜치를 다른 브랜치의 코드로 대체

```
git checkout (바꿀 브랜치)
git reset --hard (타깃 브랜치)
```

8. `git log`
   - 커밋 이력 확인

```
git log
```

   - 한 줄로 요약

```
git log --oneline
```

9. `git branch`

- 현재 설정된 브랜치 앞에 * 가 붙습니다.

```
git branch
```

- 브랜치 생성

```
git branch <브랜치명>
```