การ clone ต่างจาก checkout ใน revision control แบบ linear history เช่น SVN ตรงที่ ใน SVN repository มีที่เดียวเป็นศูนย์กลาง สิ่งที่ checkout เป็นเพียงแค่ working directory ส่วนตัวของเราแต่ใน Git การ clone เราได้ทั้งตัว repository และ working directory อันนี้ทำให้เราได้ history graph มาเป็นของเราเอง เราเรียก repository ที่ต้นทางว่า origin repository (หรือ remote repository หากอยู่ต่าง network จากเรา เช่น GitHub เองเป็นต้น) ส่วน repository ที่เรา clone มามักเรียกว่า local repositoryและจะบอกว่า Git เองก็มี checkout ซึ่งแท้แล้วก็เหมือนกับ SVN เพียงแต่ว่า Git checkout ทำได้จาก local repository มายัง working directory เท่านั้น ไม่สามารถทำได้จาก origin repository ##add, commit หากเราเปลี่ยนแปลงหลายไฟล์ใน repo ที่ clone มา จะมีแต่ไฟล์ที่ใน **Index** เท่านั้นที่สามารถบันทึกลง history เราใช้คำสั่ง add ในการเพิ่งไฟล์ลงไปใน **Index** ของ Git

จากนั้นเราก็ต้องสั่ง commit Git ก็จะไปมองหาทุกสิ่งใน **Index** แล้วบันทึกการเปลี่ยนแปลง ใน Git เราสามารถ add อย่างอื่นนอกจากไฟล์ลง **Index** ได้เช่น บรรทัดบางบรรทัดในไฟล์ (หรือ hunk)อนึ่ง การที่เรา commit ให้รู้ว่า เรา commit ไปยัง repository ที่เรา clone มาเท่านั้น และไม่มีผลไปถึง origin repository ดังนั้น…

##push หลังจากเราทำงานไปได้พักนึ่ง เราควรจะส่ง history ใหม่ๆจาก clone repo ของเรากลับไปยัง origin ที่ GitHub นี่คือตอนที่เราใช้ push push จะลอก history ของเรากลับไปยัง origin repo ##fetch, pull, merge แต่ว่าในการทำงานเป็นทีม ที่ origin repo อาจจะมีการเปลี่ยนแปลงระหว่างที่เรายังไม่ได้ push เราควรจะ fetch สิ่งที่ clone repo เราไม่มีลงมาเสียก่อน fetch จะดึงสิ่งใหม่ ๆ จาก origin มาที่ clone repository แต่ว่าจะไม่แตะต้อง working directory นี่เป็นสิ่งดีเพราะว่า เราจะได้ใช้จังหวะนี้ ดูว่ามี conflict อะไรเกิดขึ้นหรือเปล่าในสิ่งที่เรายังไม่ได้ commit ไปใน clone repository เราจะได้ resolve conflict ได้ก่อน สำหรับผู้ที่ชอบความตื่นเต้น สามารถใช้ pull ซึ่งจะดึงสิ่งใหม่ๆจาก origin ลงมา merge ทั้งบน clone repository และ working directory โดยทันที หากเป็นมี conflict จากการ merge ใน working directory เราต้อง resolve conflict นั้น ๆ ก่อนจะ commit ได้ต่อไป นั่นคือ pull แท้จริงคือ fetch ต่อเนื่องด้วย merge ในท่าเดียวนั่นเอง

##reset ทุกคนย่อมมีพลาดพลั้ง เราสามารถย้อน working directory ของเราไป ณ commit หนึ่งใดใน history ของ local repository ได้ ด้วยการ reset แต่ละ commit ใน Git จะมี id เป็นเลข hash ยาวๆเช่น 29a4c0d0549897bf2dabc90a4f7664de31d4df43 แต่เราสามารถเรียกย่อ ๆ ได้ด้วยเลขเจ็ดตัวแรก 29a4c0d

เวลาเรา reset ก็แค่บอก Git ว่าเราจะ reset ไปยัง commit id ใด และเรายังสามารถกำหนดได้ด้วยว่า จะมี path หรือ file ไหนถูก reset บ้าง (อ่านคู่มือ Git client ของตัวเองนะว่าแต่ละอันกำหนดอย่างไร)

##branch

เป็นเรื่องปกติที่ stable app จะต้องมีการแก้บั๊กหลัง roll out และในเวลาเดียวกัน เราก็อาจจะต้องเพิ่ม features ใหม่ๆที่อาจจะทำให้ app ไม่ stable สิ่งที่นิยมทำใน revisioning control คือการแยก branch ใหม่เพื่อให้เราเริ่มโค้ด features ใหม่ได้ ส่วน app ที่ stable ก็จะอยู่ใน master branch เพื่อให้งานดำเนินต่อไปได้โดยไม่ต้องรอแก้บั๊กใน master ให้เสร็จก่อน ใน Git เราใช้คำสั่ง branch เพื่อแตก branch ใหม่ออกมา Branch ใหม่ที่แยกออกมาก็จะมี history เป็นของมันเอง เราสามารถ checkout และ commit สิ่งใหม่ๆใน branch นี้ได้ นั่นคือ ก็เหมือนใน local repository หนึ่ง ๆ เราสามารถมี sub repository ย่อย ๆ นั่นเอง เพียงแต่เราเรียก sub repository พวกนี้ว่า branchพอถึงเวลาที่เราเสร็จการแก้ master branch แล้ว เราสามารถใช้คำสั่ง merge เพื่อรวมงานของ master และ new features branch เข้าด้วยกัน ไม่ต่างจากการ merge remote และ local repository ข้างต้น

##stash เวลา pull หรือ fetch เรามักจะต้อง merge เรามีการเปลี่ยนแปลงใหม่ๆใน working directory ของเราซึ่งบางทีเราก็ไม่อยากจะ merge ณ จุดนั้น เพราะจริง ๆ เรารู้ว่าเรา commit หลังจากสิ่งที่ pull หรือ fetch ก็ได้ไม่มีปัญหา ใน Git เราสามารถทำดังนั้นได้ด้วยการ stash สิ่งที่เราจะ commit เพื่อซ่อนไม่ให้ pull หรือ fetch เห็นและจะบังคับเรา merge พอเรา pull หรือ fetch เสร็จแล้ว เราสามารถดึงสิ่งที่ซ่อนไว้ใน stash กลับมาเพื่อการ commit อย่างราบรื่นต่อไป

##Fork คืออะไร Fork ไม่ใช่เรื่องของ Git โดยตรง แต่เป็นเรื่อง software development การ fork คือการเอางาน open source มาต่อยอดของเราเอง โดยที่ไม่ได้ขึ้นตรงกับแผนงานหลักของ open source นั้น ๆ ใน GitHub เรา fork ได้โดยง่าย โดยปุ่มบนเว็ป GitHub เอง (ซึ่งจริง ๆ ก็ตงจะใช้ clone ภายใน) แต่ GitHub เพิ่มความสามารถในการจัดการโปรเจคที่เรา fork มามากมายด้วย เช่นวันหนึ่งเราเกิดอยากจะ push สิ่งที่เราทำบางสิ่งกลับไปที่โปรเจค GitHub ก็มีวิธีให้ทำได้โดยง่ายเป็นระเบียบและมีบันทึกเรียบร้อย

##Git SVN Bridge คืออะไร Origin repository ของ cloned repository ไม่จำเป็นต้องเป็น Git repository origin จริง ๆอาจจะเป็น SVN repository ก็ได้ Git มี Git SVN Bridge เพื่อให้เราใช้ความสามารถของ Git ในการทำงานกับ repo ที่เป็น SVN ได้

อ้างอิงจาก https://gist.github.com/norsez/3016877