Git 치트시트

GIT CHEAT SHEET

Git은 로컬 컴퓨터에서 GitHub 관련 작업을 담당하는 무료 오픈소스 분산 버전 관리 시스템입니다. 이 치트시트는 자주 사용하는 주요 Git 명령어를 쉽게 참고할 수 있도록 정리한 것입니다.

STAGE & SNAPSHOT

스냅샷과 Git 스테이징 영역 작업

- `git status`: 작업 디렉토리에서 수정된 파일과 스테이지된 파일 상태 표시
- `git add [file]`: 현재 상태의 파일을 다음 커밋에 포함하도록 스테이지함
- `git reset [file]`: 작업 디렉토리 변경사항은 유지하며 스테이징 해제
- `git diff`: 스테이지되지 않은 변경사항 비교
- `git diff --staged`: 스테이지된 변경사항 비교 (커밋 전)
- `git commit -m [설명 메시지]`: 스테이지된 내용을 새 커밋으로 저장

SETUP

모든 로컬 저장소에서 사용할 사용자 정보 설정

- `git config --global user.name [이름]`: 버전 이력 검토 시 식별 가능한 이름 설정
- `git config --global user.email [이메일]`: 각 기록에 연동될 이메일 주소 설정
- `git config --global color.ui auto`: 명령어 결과 자동 컬러링

SETUP & INIT

사용자 정보 설정, 저장소 초기화 및 복제

- `git init`: 기존 디렉토리를 Git 저장소로 초기화
- `git clone [url]`: URL을 통해 원격 저장소 전체 복제

BRANCH & MERGE

브랜치 작업 분리, 전환 및 병합

- `git branch`: 현재 브랜치 목록 표시 (활성 브랜치 옆에 * 표시)
- `git branch [브랜치 이름]`: 현재 커밋에서 새 브랜치 생성
- `git checkout [브랜치 이름]`: 다른 브랜치로 전환
- `git merge [브랜치]`: 지정한 브랜치의 변경사항을 현재 브랜치에 병합
- `git log`: 현재 브랜치의 커밋 이력 표시

INSTALLATION & GUIS

플랫폼별 설치 및 GUI 도구

- GitHub for Windows: https://windows.github.com
- GitHub for Mac: https://mac.github.com
- Git for All Platforms (Linux, Solaris 등): http://git-scm.com

SHARE & UPDATE

원격 저장소에서 업데이트 가져오기 및 로컬 저장소 갱신

- `git remote add [별칭] [url]`: 원격 저장소 URL을 별칭으로 추가
- `git fetch [별칭]`: 원격 저장소의 모든 브랜치 가져오기
- `git merge [별칭]/[브랜치]`: 원격 브랜치를 현재 브랜치로 병합하여 최신화
- `git push [별칭] [브랜치]`: 로컬 브랜치 커밋을 원격 저장소에 전송
- `git pull`: 추적 중인 원격 브랜치의 변경사항을 가져와 병합

TRACKING PATH CHANGES

파일 삭제 및 경로 변경 버전 관리

- `git rm [file]`: 프로젝트에서 파일 삭제하고 삭제를 커밋에 준비
- `git mv [기존 경로] [새 경로]`: 파일 경로 변경 및 스테이지
- `git log --stat -M`: 경로 이동이 감지된 커밋 로그 표시

TEMPORARY COMMITS

수정한 추적 파일을 임시 저장 후 브랜치 변경

- `git stash`: 수정 및 스테이지한 변경사항 임시 저장
- `git stash list`: 임시 저장한 변경사항 목록
- `git stash pop`: 임시 저장된 변경사항 적용 및 목록에서 제거
- `git stash drop`: 임시 저장항목 삭제

REWRITE HISTORY

브랜치 수정, 커밋 업데이트 및 이력 재작성

- `git rebase [브랜치]`: 현재 브랜치 커밋을 지정 브랜치 이후로 옮겨 적용
- `git reset --hard [커밋]`: 스테이징 영역 초기화 및 작업 디렉토리를 지정 커밋 상태로 재설정

INSPECT & COMPARE

로그, 변경점, 객체 정보 확인

- `git log`: 현재 활성 브랜치의 커밋 이력
- `git log branchB..branchA`: branchB에는 없고 branchA에만 있는 커밋 표시
- `git log --follow [파일]`: 이름이 변경된 파일도 포함한 변경 커밋 표시
- `git diff branchB...branchA`: branchA에만 있는 변경사항 비교
- `git show [SHA]`: Git 객체(커밋, 태그, 파일 등) 상세 표시

IGNORING PATTERNS

원하지 않는 파일의 스테이징 및 커밋 방지

- `git config --global core.excludesfile [파일경로]`: 모든 로컬 저장소에 적용할 시스템 전역 무시 패턴 파일 설정
- `.gitignore` 파일을 생성하여 문자열 또는 와일드카드(glob) 패턴으로 제외할 파일을 지정 가능

고급 Git 워크플로우

Git Rebase 심화

- `git rebase -i HEAD~3`: 최근 3개 커밋을 대화형으로 리베이스
- `git rebase --onto main feature-branch`: 특정 브랜치를 다른 베이스로 리베이스
- `git rebase --continue`: 충돌 해결 후 리베이스 계속
- `git rebase --abort`: 리베이스 중단하고 원래 상태로 복원
- `git rebase --skip`: 현재 커밋 건너뛰기

Cherry-pick과 고급 병합

- `git cherry-pick <commit-hash>`: 특정 커밋만 현재 브랜치에 적용
- `git cherry-pick <start>..<end>`: 커밋 범위를 체리픽
- `git cherry-pick --no-commit <commit-hash>`: 커밋하지 않고 변경사항만 스테이징
- `git merge --no-ff feature-branch`: Fast-forward 병합 방지
- `git merge --squash feature-branch`: 모든 변경사항을 하나의 커밋으로 병합

Submodule 관리

- `git submodule add <repository-url> <path>`: 서브모듈 추가
- `git submodule init`: 서브모듈 초기화
- `git submodule update`: 서브모듈 업데이트
- `git submodule update --remote`: 서브모듈을 원격 저장소의 최신 커밋으로 업데이트
- `git submodule foreach git pull origin main`: 모든 서브모듈에서 pull 실행

Git Hooks와 자동화

- `pre-commit`: 커밋 전 실행되는 훅 (코드 검사, 테스트 등)
- `post-commit`: 커밋 후 실행되는 훅 (알림, 배포 등)
- `pre-push`: 푸시 전 실행되는 훅 (테스트, 린트 등)
- `commit-msg`: 커밋 메시지 검증 훅

고급 브랜치 관리

- `git branch -m old-name new-name`: 브랜치 이름 변경
- `git branch --set-upstream-to=origin/main feature-branch`: 업스트림 브랜치 설정
- `git branch --unset-upstream`: 업스트림 브랜치 설정 해제
- `git branch -d --force branch-name`: 강제로 브랜치 삭제
- `git branch --merged`: 병합된 브랜치 목록
- `git branch --no-merged`: 병합되지 않은 브랜치 목록

Git Stash 고급 사용법

- `git stash push -m "message"`: 메시지와 함께 스태시
- `git stash list`: 스태시 목록 확인
- `git stash show stash@{0}`: 특정 스태시의 변경사항 확인
- `git stash apply stash@{0}`: 스태시 적용 (스태시 유지)
- `git stash pop stash@{0}`: 스태시 적용 후 삭제
- `git stash branch new-branch stash@{0}`: 스태시에서 새 브랜치 생성

Git Bisect로 버그 찾기

- `git bisect start`: 이분 탐색 시작
- `git bisect bad <commit>`: 문제가 있는 커밋 표시
- `git bisect good <commit>`: 정상인 커밋 표시
- `git bisect run <script>`: 자동으로 이분 탐색 실행
- `git bisect reset`: 이분 탐색 종료

Git Worktree로 여러 브랜치 동시 작업

- `git worktree add ../feature-branch feature-branch`: 새 작업 트리 추가
- `git worktree list`: 작업 트리 목록
- `git worktree remove ../feature-branch`: 작업 트리 제거
- `git worktree prune`: 사용하지 않는 작업 트리 정리

Git Reflog로 복구

- `git reflog`: HEAD 변경 이력 확인
- `git reflog show branch-name`: 특정 브랜치의 변경 이력
- `git reset --hard HEAD@{2}`: reflog를 사용한 복구
- `git cherry-pick HEAD@{2}`: 특정 시점의 커밋 복구

Git Config 고급 설정

- `git config --global core.autocrlf true`: Windows에서 줄바꿈 자동 변환
- `git config --global core.safecrlf true`: 줄바꿈 변환 검사
- `git config --global pull.rebase true`: pull 시 rebase 사용
- `git config --global init.defaultBranch main`: 기본 브랜치명 설정
- `git config --global alias.co checkout`: 명령어 별칭 설정
- `git config --global alias.br branch`: 브랜치 명령어 별칭
- `git config --global alias.ci commit`: 커밋 명령어 별칭

Git LFS (Large File Storage)

- `git lfs install`: Git LFS 초기화
- `git lfs track "*.psd"`: 특정 파일 타입을 LFS로 추적
- `git lfs track "*.zip"`: 압축 파일을 LFS로 추적
- `git lfs ls-files`: LFS로 추적되는 파일 목록
- `git lfs pull`: LFS 파일 다운로드

Git Archive로 배포 패키지 생성

- `git archive --format=zip --output=release.zip main`: ZIP 형식으로 아카이브
- `git archive --format=tar.gz --output=release.tar.gz main`: TAR.GZ 형식으로 아카이브
- `git archive --format=zip --prefix=project/ main`: 접두사와 함께 아카이브

고급 Git 명령어 조합

```
# 최근 커밋들의 통계
git log --oneline --stat -10

# 특정 파일의 변경 이력
git log --follow -- filename

# 브랜치별 커밋 수 비교
git rev-list --count main
git rev-list --count feature-branch

# 특정 기간의 커밋
git log --since="2023-01-01" --
until="2023-12-31"

# 특정 작성자의 커밋
git log --author="John Doe"

# 커밋 메시지로 검색
git log --grep="bug fix"

# 파일 변경 내용과 함께 보기
git log -p filename

# 그래프로 브랜치 히스토리 보기
git log --graph --oneline --all
```