WF Characterisation

Rules				Git Workflows				
				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
1. Setup [1]	a) Repository creation	A) Clone					REC	
		B) GitHub-style Fork						
	b) Branch role assignment [3]	i. Main [4]	A) Master	✓	✓	✓	✓	✓
			B) Develop			✓		✓
			C) Feature					
			D) Topic					
			E) Production					
			F) Release					✓
		ii. Integration	A) Master			✓		✓
			B) Develop			✓		✓
			C) Feature					✓
			D) Topic					✓
			E) Production					✓
			F) Release					✓
			G) Fix					✓
			H) HotFix					✓
			I) BugFix					✓
		iii. Change	A) Master		✓		✓	✓
			B) Develop	✓	✓		✓	✓
			C) Feature	✓	✓	✓	✓	✓
			D) Topic	✓	✓	✓	✓	✓
			E) Production	✓	✓		✓	✓
			F) Release	✓	✓		✓	✓
			G) Fix	✓	✓		✓	✓
			H) HotFix	✓	✓		✓	✓
			I) BugFix	✓	✓		✓	✓
		iv. Release	A) Master	✓		✓	✓	✓
			B) Develop					
			C) Feature					
			D) Topic					
			E) Production					
			F) Release			✓	✓	✓
			G) Fix					✓
			H) HotFix					
			I) BugFix					
		v. Fix	A) Master					
			B) Develop	✓				
			C) Feature	✓				
			D) Topic	✓				
			E) Production	✓				
			F) Release	✓				
			G) Fix	✓	✓	✓		
			H) HotFix	✓		✓		
			I) BugFix	✓		✓		
	c) Development	i. Main	A) Central repository			REQ		
			B) Local repository					
			C) Forked repository					
		ii. Integration	A) Central repository			REQ		
			B) Local repository					
			C) Forked repository					
			A) Central repository					

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

Rules				Git Workflows				
				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
	c) Development environment	iii. Change	B) Local repository	REC	REQ	REC	REC	REQ
			C) Forked repository					
		iv. Release	A) Central repository					
			B) Local repository					
			C) Forked repository					
		v. Fix	A) Central repository					
			B) Local repository	REC				
			C) Forked repository					
2. Branching strategy [5]	a) Creation event [6]	ii. Integration [7]	A) To work on a new feature					
			B) To create a fix					
			C) To prepare a release					
			D) To integrate work			REQ		
		iii. Change [8]	A) To work on a new feature	REQ	REQ	REQ	REC	REQ
			B) To create a fix					
			C) To prepare a release					
			D) To integrate work					
		iv. Release [9]	A) To work on a new feature					
			B) To create a fix					
			C) To prepare a release			REQ	REC	REQ
			D) To integrate work					
		v. Fix [10]	A) To work on a new feature					
			B) To create a fix	REQ		REQ		
			C) To prepare a release					
			D) To integrate work					
	b) Parent branch [11]	ii. Integration	i. Main			REQ		
			iii. Change					
			iv. Release					
		iii. Change	i. Main	REQ	REQ	REQ	REQ	REQ
			ii. Integration			REC		
			iv. Release			PRO		
		iv. Release	i. Main				REQ	REQ
			ii. Integration			REC		
			iii. Change					
		v. Fix	i. Main			REC		
			ii. Integration					
			iii. Change					
			iv. Release	REQ				
	c) Broken code is not allowed	i. Main		REQ	REQ	REQ	REQ	
		ii. Integration						
		iii. Change					REQ	
		iv. Release		REQ			REQ	
		v. Fix						
	d) Production-ready [12]	i. Main		REQ		REQ	REQ	
		ii. Integration						
		iii. Change					REQ	
		iv. Release		REQ			REQ	
		v. Fix						
		i. Main	A) Short-lived					
			B) Long-lived			REC	REC	REQ
		ii. Integration	A) Short-lived					
			B) Long-lived			REC		

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

Rules				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
	e) Lifetime	iii. Change	A) Short-lived		REC	REC	REC	REC
			B) Long-lived					
		iv. Release	A) Short-lived			REC	REC	
			B) Long-lived					
		v. Fix	A) Short-lived			REC		
			B) Long-lived					
	f) Delete branch after [13]	i. Main	A) Integration completed					
			B) Additional tasks completed					
		ii. Integration	A) Integration completed					
			B) Additional tasks completed					
		iii. Change	A) Integration completed			REC	REC	REQ
			B) Additional tasks completed					
		iv. Release	A) Integration completed			REC		
			B) Additional tasks completed			REC		
		v. Fix	A) Integration completed			REC		
			B) Additional tasks completed					
	a) Code integration event [15]	i. Main	A) End of working day					
			B) Every <n> weeks					
			C) When work completed					
			D) After rebase					
			E) After code review completed					
			F) After tests passing in Dev Env.					
			G) After tests passing in Test/QA/Stage					
			H) After tests passing in Production					
			I) To update the target branch					
			J) As frequently as possible					REC
		ii. Integration	A) End of working day					
			B) Every <n> weeks					
			C) When work completed					
			D) After rebase					
			E) After code review completed					
			F) After tests passing in Dev Env.					
			G) After tests passing in Test/QA/Stage					
			H) After tests passing in Production					
			I) To update the target branch					
			J) As frequently as possible					
		iii. Change	A) End of working day					
			B) Every <n> weeks					
			C) When work completed			REQ		
			D) After rebase					
			E) After complete code review	REC	REC		REQ	REC
			F) After tests passing in Dev Env.	REC			REQ	
			G) After tests passing in Test/QA/Stage					
			H) After tests passing in Production	REC				
			I) To update the target branch					
			J) As frequently as possible	REQ			REC	
		iv. Release	A) End of working day					
			B) Every <n> weeks					
			C) When work completed					
			D) After rebase					
			E) After code review completed			REQ		

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

Rules				Git Workflows				
				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
3. Code integration strategy [14]		iv. Release	F) After tests passing in Dev Env.					
			G) After tests passing in Test/QA/Stage					
			H) After tests passing in Production					
			I) To update the target branch					
			J) As frequently as possible					
		v. Fix	A) End of working day					
			B) Every <n> weeks					
			C) When work completed			REQ		
			D) After rebase					
			E) After code review completed	REC				
			F) After tests passing in Dev Env.	REC				
			G) After tests passing in Test/QA/Stage					
			H) After tests passing in Production	REC				
			I) To update the target branch					
			J) As frequently as possible					
	b) Main integrated into	ii. Integration	A) Upstream					
			B) Downstream					
		iii. Change	A) Upstream				REC	
			B) Downstream					REQ
		iv. Release	A) Upstream					
			B) Downstream					
		v. Fix	A) Upstream					
			B) Downstream					
	c) Integration integrated into [16]	i. Main	A) Upstream					
			B) Downstream					
		iii. Change	A) Upstream					
			B) Downstream					
		iv. Release	A) Upstream					
			B) Downstream					
		v. Fix	A) Upstream					
			B) Downstream					
	d) Change integrated into [17]	i. Main	A) Upstream	REQ	REQ		REQ	REQ
			B) Downstream					
		ii. Integration	A) Upstream			REQ		
			B) Downstream					
		iii. Change	A) Upstream		REC			
			B) Downstream					
		iv. Release	A) Upstream	REQ				
			B) Downstream					
		v. Fix	A) Upstream					
			B) Downstream					
	e) Release integrated into [18]	i. Main	A) Upstream			REQ		REC
			B) Downstream					
		ii. Integration	A) Upstream			REQ		
			B) Downstream					
		iii. Change	A) Upstream					
			B) Downstream					
		v. Fix	A) Upstream					
			B) Downstream					
		i. Main	A) Upstream	REQ		REQ		
			B) Downstream					

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

Rules				Git Workflows				
				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
	f) Fix integrated into [19]	ii. Integration	A) Upstream			REQ		
			B) Downstream					
		iii. Change	A) Upstream	REQ				
			B) Downstream					
		iv. Release	A) Upstream			REC		
			B) Downstream					
	g) Code integration mechanism [20]	i. Main	A) fast-forward merge					PRO
			B) non-fast forward merge					PRO
			C) rebase					REQ
			D) cherry-pick					REQ
			E) fetch with rebase and merge					PRO
			F) fetch and merge					PRO
		ii. Integration	A) fast-forward merge					
			B) non-fast forward merge					
			C) rebase					
			D) cherry-pick					
			E) fetch with rebase and merge					
			F) fetch and merge					
		iii. Change	A) fast-forward merge				REC	PRO
			B) non-fast forward merge			REQ	REC	PRO
			C) rebase		REC			REQ
			D) cherry-pick					REQ
			E) fetch with rebase and merge					PRO
			F) fetch and merge					PRO
		iv. Release	A) fast-forward merge					PRO
			B) non-fast forward merge			REQ		PRO
			C) rebase					REQ
			D) cherry-pick				REC	REQ
			E) fetch with rebase and merge					PRO
			F) fetch and merge					PRO
		v. Fix	A) fast-forward merge					
			B) non-fast forward merge			REQ		
			C) rebase					
			D) cherry-pick					
			E) fetch with rebase and merge					
			F) fetch and merge					
	h) Merge options [21]	A) force						
		B) force-with-lease						
		C) no options						
	i) Code review approach before integrating from [22]	i. Main	A) Staged					
			B) At central repository					
			C) Through pull/merge request					
		ii. Integration	A) Staged					
			B) At central repository					
			C) Through pull/merge request					
		iii. Change	A) Staged		REC		REC	REC
			B) At central repository					
			C) Through pull/merge request	REC	REC			
		iv. Release	A) Staged					
			B) At central repository					
			C) Through pull/merge request					

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

Rules				Git Workflows				
				GitHub Flow	Feature Branch	GitFlow	Trunk-based Development	Cactus Model
		v. Fix	A) Staged					
			B) At central repository					
			C) Through pull/merge request	REC				
4. Development conventions [23]	a) Commit message guidelines [24]	A) GitHub convention						
	b) Naming convention [25]	i. Main	A) semantic versioning (semver)					
		ii. Integration	A) semantic versioning (semver)					
			B) develop			REQ		
		iii. Change	A) semantic versioning (semver)					
		iv. Release	A) semantic versioning (semver)			REC		
			B) release-*			REQ		
		v. Fix	A) semantic versioning (semver)			REC		
			B) hotfix-* bugfix-* fix-*			REQ		
	c) Tags used [26]	i. Main				REQ		
		ii. Integration						
		iii. Change						
		iv. Release						REC
		v. Fix						
	d) Version bump used [27]					REQ		

Types of guidelines		
Key	Label	Description
REQ	Requirement	Guideline enforced by the workflow, non-optional (MUST, SHALL)
REC	Recommendation	Suggested guideline that if ignored does not affect the main purpose of the worflow, optional (SHOULD)
PRO	Prohibition	Guideline that is explicitly forbidden (MUST NOT, SHALL NOT)

[1] What steps are needed to prepare the project for the development process from the contributor point of view? (e.g., how to setup the initial working copy to start contributing to the project?)

[2] What mechanism (clone or GitHub fork) is used to create the working copy for a contributor to the project?

[3] In the workflow, which branch assumes the role of each these types of branches?

[4] Contains all the changes associated with the last stable release (other names: Master, Trunk)

[5] Which conventions are followed to decide on how branches should be used to align with the conventions, requirements and objectives of the development project? (e.g., what are the types of branches used along the development process?)

[6] When a contributor should create a new branch?

[7] Allows the integration of multiple changes that are not yet ready to be released (other names: Develop)

[8] All work associated with the development of a new feature (other names: Feature, topic)

[9] Once the changes are completed, tested and integrated they can be included in this branch, which contains code that is ready for production (other names: Production)

[10] Include development work to correct detected bugs and other emergency fixes (other names: HotFix, BugFix)

[11] From which branch the contributors should branch off?

[12] Should the branch be at all times ready for production (i.e., containing only code that compiles, with all tests passing, and fully integrated)?

[13] Under what condition(s) should the branch be removed?

[14] What approach or mechanism is followed to merge branches back? (e.g., what triggers a merge operation in the development process and which branches participate in it?)

[15] At which point a branch should be merged back (e.g., when the work is ready for production, when a rebase has been applied, when the work in the feature is completed, every <n> number of weeks)?

[16] Which branch should a Working branch be merged back to?

[17] Which branch should a Change branch be merged back to?

[18] Which branch should a Release branch be merged back to?

[19] Which branch should a Fix branch be merged back to?

[20] What mechanism (e.g., fast-forward merge, non-fast-forward merge, rebase, cherry-pick, fetch with rebase and merge, fetch and merge) should be used for merging a branch back?

[21] How changes that are intended to be merged should be pushed to the remote repository (e.g., with force option, with force-with-lease option, without options)?

[22] Should pull requests be used before merging back a branch?

[23] What additional guidelines are requested from contributors to the development project that are not specifically related to any of the previous categories? (e.g., what naming conventions are followed for the final release of the software product?)

[24] Which naming convention (e.g., GitHub or other) should be followed for commit messages?

[25] Which naming convention should be followed for a branch?

[26] Are tags used to identify a branch?

[27] Is version bump used to mark the creation of a new version?